Characteristics and regularities of graphs and words

Christou, Michalis

Awarding institution:
King's College London

Characteristics and regularities of graphs and words

Michalis Christou

A thesis submitted to the University of London in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Department of Informatics

King's College, University of London

September 2013

Abstract

In the last years there has been an increasing interest in discrete structures such as graphs, trees and strings that appear in Mathematics and Computer Science but also in many other interdisciplinary areas including bioinformatics, pattern matching and data compression. We investigate several types of regularities and characteristics appearing in these structures, as well as algorithms for their computation.

Graphs are probably the most popular objects of study in discrete mathematics. We show some progress in extremal graph theory, i.e. problems which investigate extremal graphs satisfying certain properties (maximizing planar graphs under girth constraints, degree/diameter problem for trees and pseudo trees, bipartite Ramsey numbers involving stars, stripes and trees).

Words, also called strings, are also structures that have acquired great importance in recent years mainly due to their use in DNA modelling. We show some new results regarding the identification and appearance of seeds in strings (a linear time algorithm that computes the minimal left-seed array of a given string, an $O(n \log n)$ time algorithm that computes the minimal right-seed array of a given string x, a linear-time solution to compute the maximal left-seed/right-seed array, the appearance of quasiperiodic structures in Fibonacci words and general words). We also show some progress regarding the identification and appearance of abelian regularities in strings (quadratic time algorithms for the identification of all abelian periods of a string, a linear time algorithm for the identification of all abelian borders of a string, bounds on the number of abelian borders a word and bounds on the length of the shortest border of a binary string). Furthermore, we investigate the average number of

regularities in a word and we are reveal some interesting properties of Padovan words, a family of words related to the Fibonacci sequence. Finally we present a problem on trees: how to output all subtree repeats of a tree in linear time using a string representation of the tree.

Keywords: algorithms, combinatorics, graphs, trees, strings, periodicity, extremal

Acknowledgements

Thanks are extended to my family which has always been close to me and to my supervisors, Prof Costas Iliopoulos and Prof Maxime Crochemore, for their continuous supervision and guidance throughout the completion of this PhD thesis, as well as to my various collaborators (Manolis Christodoulakis, Tomáš Flouri, Ondřej Guth, Jan Janoušek, Marcin Kubica, Bořivoj Melichar, Mirka Miller, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Bartosz Szreder, Tomasz Waleń and Jan Žďárek).

Disclaimer

I declare the following to be my own work, unless otherwise referenced, as defined by the University's policy on plagiarism. The following articles have been accepted for publication, as part of my PhD research, and some parts are reproduced in this work:

- Manolis Christodoulakis, Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. Abelian borders in binary words. Discrete Applied Mathematics, 171: 141-146, 2014.

- Manolis Christodoulakis, Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. On the average number of regularities in a word. Theoretical Computer Science, 525: 3-9, 2014.

- Michalis Christou, Maxime Crochemore, Tomáš Flouri, and Costas S. Iliopoulos, Jan Janoušek, Solon P. Pissis and Jan Žďárek . Tree Template Matching in Unranked Ordered Trees, Journal of Discrete Algorithms, 20: 51-60, 2013.

- Michalis Christou, Costas S. Iliopoulos and Mirka Miller. Bipartite Ramsey numbers involving stars, stripes and trees. Electronic Journal of Graph Theory and Applications, 1 (2): 89-99, 2013.

- Manolis Christodoulakis, Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. On the appearance of seeds in words. Journal of Combinatorial Mathematics and Combinatorial Computing, 2013. (accepted)

- Manolis Christodoulakis and Michalis Christou. Abelian concepts in words-a review. Festschrift for Bořivoj Melichar, Prague Stringology Conference, 2012.

- Michalis Christou, Costas S. Iliopoulos and Mirka Miller. Degree/ diameter problem for trees and pseudotrees. AKCE International

Journal of Graphs and Combinatorics, 10 (4): 377-389, 2013. (accepted)

- Manolis Christodoulakis, Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. Abelian borders in words. Fundamenta Informaticae, 2013. (accepted)

- Manolis Christodoulakis, Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. Overlapping factors in words. Australasian Journal of Combinatorics, 57: 49-64, 2013.

- Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. Quasiperiodicities in Fibonacci strings, Ars Combinatoria, 2012. (accepted)

- Michalis Christou, Maxime Crochemore, Tomáš Flouri, and Costas S. Iliopoulos, Jan Janoušek, Bořivoj Melichar and Solon P. Pissis. Computing all subtree repeats in ordered trees.Information Processing Letters, 112(24):958-962, 2012.

- Michalis Christou, Costas S. Iliopoulos and Mirka Miller. Maximizing the size of planar graphs under girth constraints. Journal of Combinatorial Mathematics and Combinatorial Computing, 2012. (accepted)

- Michalis Christou, Maxime Crochemore and Costas S. Iliopoulos. Identifying all abelian periods of a string in quadratic time and relevant problems. International Journal of Foundations of Computer Science, 23 (6): 1371-1384, 2012.

- Michalis Christou, Maxime Crochemore, Ondřej Guth, Costas S. Iliopoulos and Solon P. Pissis. On the left and right seeds of a string. Journal of Discrete Algorithms, 17: 31-44, 2012. (accepted)

- Michalis Christou, Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Solon P. Pissis, Jakub Radoszewski, and Wojciech Rytter, B. Szreder and Tomasz Waleń. Efficient seed computation revisited. Theoretical Computer Science, 483: 171-181, 2013.

- Michalis Christou, Maxime Crochemore, and Costas S. Iliopoulos. Quasiperiodicities in fibonacci strings. In Local Proceedings of International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2012), pages 13-22, 2012.

- Michalis Christou, Maxime Crochemore, Ondřej Guth, Costas S. Iliopoulos and Solon P. Pissis. On the right-seed array of a string. In Proceedings of the 17th annual international Computing and Combinatorics conference, COCOON'11, pages 492-502, 2011. Springer -Verlag.

- Michalis Christou, Maxime Crochemore, Tomáš Flouri, Costas S. Iliopoulos, Jan Janoušek, Bořivoj Melichar and Solon P. Pissis. Computing all subtree repeats in ordered ranked trees. In Roberto Grossi, Fabrizio Sebastiani and Fabrizio Silvestri, editors, String Processing and Information Retrieval, volume 7024 of Lecture Notes in Computer Science, pages 338-343, 2011. Springer Berlin / Heidelberg.

- Michalis Christou, Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Bartosz Szreder and Tomasz Waleń. Efficient seed computation revisited. In Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM 2011), volume 6661 of Lecture Notes in Computer Science, pages 350-363, 2011. Springer.

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Discrete structures such as graphs and words are an essential element in the field of Computer Science. These objects, also being studied in the field of Discrete Mathematics, may appear as huge data structures in several fields such as data compression and bioinformatics (see [184, 185]).

As a consequence, there is a lot of interest about algorithms that identify any form of regularity in these structures. An example is the modelling of the DNA strand in the area of Computational Biology as a word on a four letter alphabet (the letters A, C, G and T stand for the nucleobases adenine, cytosine, guanine and thymine respectively). Repeats in the DNA are of great importance in Biology and strongly related to periodicities in words. For example squares, two identical subwords next to each other, in DNA help to determine inherited characteristics and thus are useful in determining parentage. Furthermore songs can be considered as large words composed of notes and algorithms relevant to regularities in words can be used in several applications, e.g. in song identification services. Additionally finding repeats in words is crucial in compressing them and hence pictures or other type of data that are stored using words can be stored more efficiently. Trees, a special type of graphs is used to represent hierarchical data and have many applications in biology (phylogenetic trees). A graph is the easiest way to represent a network, such as the world wide web, road networks, social networks, public utility networks, logistical networks etc. Characteristics of graphs are often employed in order to construct efficient networks or identify structures of interest within them.

Words, also called strings, appear in many areas of Mathematics and Computer Science as well as in several interdisciplinary areas. Some fundamental periodicities in a word include the runs and powers occurring in it, such as squares and cubes. Apart from algorithmic interest, in the last years a lot of research has been done on bounds on the maximum

number of distinct periodicities in a word. These bounds are essential elements of the analysis of some algorithms on words.

This thesis contains results on characteristics and regularities in words. Some computational results are shown like algorithms for the computation of seed arrays and algorithms for the identification of the overlapping factors of word.

The frequency of appearance of regularities in words is also investigated. The maximum number of distinct seeds in a word is found to be between $\frac{1}{6}n^2 + o(n^2)$ and $\frac{1}{4}n^2 + o(n^2)$ and the maximum number of distinct overlapping factors in a word is also studied. Furthermore, some bounds for the average number of powers with exponent r in a word are given, which are then extended for runs and palindromes as well as for some of their abelian varieties (abelian squares and cubes). Finally it is proved that a word has $O(n)$ seeds on average.

The appearance of abelian regularities in words is also investigated. Several algorithms for the computation of all abelian periods of a string x, for the computation of all weak abelian periods of a word and for the computation of all abelian borders of a word x are given. It is proved that the average length of abelian borders of a word x is $\frac{n}{2}$, if it exists, and that a binary word of length n has $\Theta(\sqrt{n})$ abelian borders on average. Finally, we investigate the number of binary words whose shortest abelian border has a given length, by identifying relations with Dyck words, and we give some bounds on the number of abelian border-free words of a given length.

The appearance of regularities in special words is also investigated. We identify all overlapping factors, left/right seeds, covers, seeds and abelian borders of a Fibonacci string, as well as all covers of a circular Fibonacci string. We give further comments on the number of distinct seeds in Fibonacci strings. We identify all borders of a Padovan word,

we give an elementary analysis of factors of the Padovan word, we identify all covers of Padovan words and we give some comments regarding the squares and cubes in a Padovan word. We also comment on the appearance of abelian borders in Thue-Morse words. Finally a string representation of a tree is used to identify all the repeating subtrees of a tree with n nodes in $O(n)$ time.

Graphs arise in many areas of mathematics and computer science, having applications in many other fields as well. Extremal graph theory problems usually ask for the maximum or minimum size or order of a graph having certain characteristics. Such questions are often quite natural in the construction of networks or circuits. Throughout this thesis we investigate extremal graph theory problems such as the degree/diameter problem, the EX-problem and the Ramsey problem on special types of graphs like planar graphs, trees and special types of trees and bipartite graphs. Some extremal solutions and bounds are given, as well as some constructions.

The rest of the thesis is structured as follows. Firstly, in Part II we give the mathematical background and the terminology used throughout this thesis. Next, in Part III we show some results regarding regularities in words and in Part IV we analyse some extremal graph theory problems on special types of graphs. Finally we give some concluding remarks and open problems in Part V together with some helpful comments in Part VI.

Part II

Mathematical Background

Chapter 1

Words

1.1 Basic definitions

We define an *alphabet* Σ as a finite, non-empty set of symbols. An ordering can be defined via a bijection $\phi : \Sigma \to \{1, 2, \ldots, \sigma\}$, where $|\Sigma| = \sigma$. A *ranked alphabet* is a couple $\mathcal{A} = (\Sigma, \varphi)$, where Σ is a finite, non-empty set of symbols and φ is a mapping $\varphi : \Sigma \mapsto \mathbb{N}$.

Throughout this thesis we consider a word, also called string, x of length $|x| = n$, $n \geq 0$, on a fixed alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$. The word with zero symbols is denoted by ε. The set of all words over the alphabet Σ is denoted by Σ^*. A word x of length n is represented by $x_1 x_2 \ldots x_n$, where $x_i \in \Sigma$ for $1 \leq i \leq n$. Whenever x is a non-empty word it is represented as $x[1\,..\,n]$. If u and v are words then uv is the word obtained by concatenating the symbols of v after the symbols of u. A word w is a *factor* (also called *substring* or *subword*) of x if $x = uwv$ for two words u and v; it is a *prefix* of x if u is empty and a *suffix* of x if v is empty. A *proper* factor of x is a factor which is not equal to x itself; *proper* prefixes, suffixes and borders are defined similarly. We denote the longest common prefix of two words x and y as $LCP(x, y)$.

1.2 Periodicity

A word u is a *border* of x if u is both a proper prefix and a suffix of x. *The border* of x, denoted by $Border(x)$, is the length of the longest border of x. However, a word is never a border of itself, i.e. $Border(x) < n$. A non-empty word u is a *period* of x if x is a prefix of u^k for some positive integer k (x^k is the concatenation of k copies of x), or equivalently if x is a prefix of ux. The length of u is also called a period (or an integer period) of x. *The period* of x, denoted by $Period(x)$, is the length of the shortest period of x.

As follows from the above definitions, for any string x, $Period(x) +$ Border$(x) = |x|$. The exponent of x is the rational ratio $|x|/Period(x)$.

A word w is a *square* of x if it is a factor of x and $w = yy$ for some non-empty word y. A word w is a *cube* of x if it is a factor of x and $w = yyy$ for some non-empty word y. More generally a word w is an *r-power* of x if it is a factor of x and $w = y^r$, for some non empty word y, and $r \in \{2, 3, 4, \dots\}$.

A non-empty word w is a *palindrome* of x if it is a factor of x and $w[i] = w[|w| + 1 - i]$ for $1 \leq i \leq \lfloor \frac{w+1}{2} \rfloor$.

A *run* is a maximal (non-extendible) occurrence of a repetition of rational exponent at least two. This means that the factor $x[i\,..\,j]$ is a run if it has the following three properties:

- $x[i\,..\,j]$ has period p and $j - i + 1 \geq 2p$
- $x[i-1] \neq x[i+p-1]$ (if $x[i-1]$ is defined), $x[j+1] \neq x[j-p+1]$ (if $x[j+1]$ is defined)
- $x[i..i+p-1]$ is primitive, that is, it is not a proper integer power (2 or larger) of another word.

1.3 Quasiperiodicity

For two words $u = u[1\,..\,m]$ and $v = v[1\,..\,n]$ where a suffix of u equals a prefix of v, $u[m-\ell+1\,..\,m] = v[1\,..\,\ell]$ for some $1 \leq \ell \leq m$, the word $u[1\,..\,m]v[\ell+1\,..\,n] = u[1\,..\,m-\ell]v[1\,..\,n]$ is called a *superposition* of u and v with an *overlap* of length ℓ.

A word w is a *quasiperiodic square* of x if it is a factor of x and $w = yv = uy$, where y and v are non-empty words and $|y| > |v|$. In this case, the factor y is called an *overlapping factor* of x.

A word y of length m is a *cover* of x if both $m < n$ and there exists a set of positions $P \subseteq \{1, \ldots, n-m+1\}$ that satisfies both $x[i\,..\,i+m-1] = y$ for all $i \in P$ and $\bigcup_{i \in P}\{i, \ldots, i+m-1\} = \{1, \ldots, n\}$. A word x is *superprimitive* if its only cover is itself, otherwise it is *quasiperiodic*. A word v is a *seed* of x, if it is a cover of a superword of x, where a superword of x is a word of form uxv and u, v are possibly empty words. A *left seed* of a word x is a prefix of x that is a cover of a superword of x of the form xv, where v is a possibly empty word. Similarly, a *right seed* of a word x is a suffix of x that is a cover of a superword of x of the form vx, where v is a possibly empty word.

The following example shows all left seeds, right seeds, covers and seeds of the word $F_6 =$ `abaababaabaab` and Figure 1.1 illustrates that *aba* is the shortest seed of F_6.

Figure 1.1: *aba* is the shortest seed of *abaababaabaab*

Example

Covers of F_6	abaab, abaababaabaab
Left seeds of F_6	aba, abaab, abaaba, abaababa, abaababaa, abaababaab, abaababaaba, abaababaabaa, abaababaabaab
Right seeds of F_6	abaab, abaabaab, babaabaab, ababaabaab, aababaabaab, baababaabaab, abaababaabaab
Seeds of F_6	aba, abaab, abaaba, abaababa, abaababaa, abaababaab, abaababaaba, abaababaabaa, abaababaabaab

1.4 Abelian concepts

Definitions relative to Parikh vectors are as in [74, 114]. The *Parikh vector* of a string x, denoted by $\mathcal{P}_x$, enumerates the number of times each letter of Σ occurs in x. That is $\mathcal{P}_x[i]$ is the number of occurrences of a_i in x, where $1 \leq i \leq \sigma$. The sum of the components of a Parikh vector is denoted by $|\mathcal{P}|$. Given two Parikh vectors $\mathcal{P}, \mathcal{Q}$ we write $\mathcal{P} \subseteq \mathcal{Q}$ if $\mathcal{P}[i] \leq \mathcal{Q}[i]$ for every $1 \leq i \leq \sigma$. The string x is said to have an *abelian period* (h, p) if $x = u_0u_1...u_{k-1}u_k$ such that:

$$\mathcal{P}_{u_0} \subseteq \mathcal{P}_{u_1} = \cdots = \mathcal{P}_{u_{k-1}} \supseteq \mathcal{P}_{u_k}, \quad |\mathcal{P}_{u_0}| = h \text{ and } |\mathcal{P}_{u_1}| = p$$

Factors u_0 and u_k are called the *head* and the *tail* of the abelian period respectively. Moreover, x is said to have a *weak abelian period* p if $|\mathcal{P}_{u_0}| = |\mathcal{P}_{u_1}| = p$.

A string y of length $|y| = m < n$ is an *abelian border* of x if $\mathcal{P}_y = \mathcal{P}_{x[1..m]} = \mathcal{P}_{x[n-m+1..n]}$. A string that has only the empty abelian border is called an *abelian border-free* string.

Figure 1.2: All abelian borders of the string $x = caabbacabbca$

Example String $x = caabbacabbca$ has abelian borders of length $2, 5, 7$ and 10, as shown in Figure 1.2.

Example String $x = aabacaabcacba$ has $(2, 5)$ as an abelian period and 5 as a weak abelian period as shown in the figure below.

Figure 1.3: $(2, 5)$ is an abelian period of x and 5 is a weak abelian period of x, where $x = caabbacabbca$

A natural order can be defined on abelian periods as follows: let (h, p) and (h', p') be abelian periods of a word y, then $(h, p) < (h', p')$ if $p < p'$ or ($p = p'$ and $h < h'$).

An *abelian square*, also called a *weak repetition*, is a word of form yz, where y and z are non-empty words having the same letter composition (same number of occurrences of each letter). The definition extends to other powers.

1.5 Special words

Sturmian words are infinite words over a binary alphabet that have exactly $n + 1$ factors of length n for each $n \geq 0$. They form a large family of words, including the Fibonacci words described below.

The n^{th} *Fibonacci number* denoted by f_n is defined by the following recurrence and initial conditions:

$$f_0 = 1, \quad f_1 = 1, \quad f_n = f_{n-1} + f_{n-2} \quad n \in \{2, 3, 4, \dots\}$$

The first few terms are: $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ We define a (finite) *Fibonacci string* F_k , $k \in \{0, 1, 2, \ldots\}$, as follows:

$$F_0 = b, \quad F_1 = a, \quad F_n = F_{n-1}F_{n-2} \quad n \in \{2, 3, 4, \ldots\}$$

Notice that $|F_n| = f_n$, the n^{th} Fibonacci number. A (finite) *circular Fibonacci string* $C(F_k)$, $k \in \{0, 1, 2, \ldots\}$, is made by concatenating the first letter of F_k to its last letter. The *infinite Fibonacci word* is the infinite word $F_\infty = abaababaabaababaababa\ldots$ which has every Fibonacci word except F_0 as a prefix.

F_0	b
F_1	a
F_2	ab
F_3	aba
F_4	abaab
F_5	abaababa
F_6	abaababaabaab
F_7	abaababaabaababaababa

Figure 1.4: The first eight Fibonacci strings

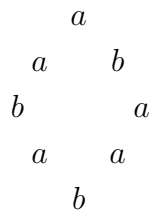


Figure 1.5: $C(F_5)$

The Thue-Morse words are binary words defined as follows:
$T_0 = 0$, $T_n = T_{n-1}\overline{T_{n-1}}$, where $n \in \mathbb{Z}^+$ and $\overline{T_{n-1}}$ is the bitwise negation of T_{n-1}.

The Thue-Morse words can be also generated from the substitution map m where:

$$m(0) = 01 \text{ and } m(1) = 10,$$

then $T_0 = 0$ and $T_n = m(T_{n-1}[1])m(T_{n-1}[2])\ldots m(T_{n-1}[|T_{n-1}|])$, for $n \in \mathbb{Z}^+$.

T_0	0
T_1	01
T_2	0110
T_3	01101001
T_4	0110100110010110
T_5	01101001100101101001011001101001
T_6	0110100110010110100101100110100110010110011010010110100110010110

Figure 1.6: The first seven Thue-Morse words

A *hole* (also called a *do not know* or a *do not care* symbol or a *wild card*) is a symbol (usually represented by a .) that is one of the letters from our alphabet Σ but we don know which (not necessarily the same each time). A *partial word* is a word that may contain some holes. More formally a partial word is a partial function $f : 1, 2, \ldots, n \to \Sigma$. At places where f is not specified we have a hole.

Chapter 2

Graphs

2.1 Basic definitions

We consider an undirected graph $G(V, E)$, where V is the set of *vertices*, also called *nodes*, and E is the set of *edges*. The *complement* graph $\overline{G}(V, \overline{E})$ of G has the same vertices as G but edges that appear in G do not appear in $\overline{G}$ and edges that do not appear in G appear in $\overline{G}$. The *order* of a graph is the number of its vertices. The *size* of a graph is the number of its edges. A *path* $P_n = P_n(V, E)$ is a graph with $V = \{x_1, x_2, \ldots, x_n\}$ and $E = \{x_1x_2, x_2x_3, \ldots, x_{n-1}x_n\}$. The end vertices of P_n are x_1 and x_n and the *length* of P_n is equal to $n - 1$. A graph is *connected* if there is a path from any node of the graph to any other node in the graph. In the case that two nodes of the graph are not connected with a path the graph is *disconnected*. The *diameter* of a graph is the length of a longest shortest path between any two vertices of the graph. The *eccentricity* $\epsilon(v)$ of a vertex v in a connected graph G is the maximum distance between v and any other vertex u of G, where the *distance* between two vertices is the length of a shortest path that connects them (in the case that there is no such path their distance is infinite). For a disconnected graph, all vertices are defined to have infinite eccentricity. Note that the

maximum eccentricity over all vertices of a graph is the diameter of the graph. The minimum eccentricity over all the vertices of the graph is called the *radius* of the graph. A *cycle* $C_n = C_n(V, E)(n \geq 3)$ is a graph with $V = \{x_1, x_2, \ldots, x_n\}$ and $E = \{x_1x_2, x_2x_3, \ldots, x_{n-1}x_n, x_nx_1\}$. The *length* of C_n is equal to n. A *cycle* is called *odd/even* if its length is *odd/even*. The *girth* $g = g(G)$ of a graph G is the length of its shortest cycle. A graph containing no cycles is called an *acyclic* graph. The *degree* of a vertex $v \in G$ is denoted by $d(v)$ and is equal to the number of vertices to which v is joined by an edge.

A *subdivision* of a graph G is a graph resulting from the subdivision of edges in G (H is also called a *subdivided* graph G), where the *subdivision* of some edge e with endpoints $\{u, v\}$ yields a graph containing one new vertex w, and two new edges $\{u, w\}$ and $\{w, v\}$ replacing e.

A *vertex colouring* of a graph is a labelling of the graph's vertices with colours such that no two adjacent vertices have the same colour.

2.2 Special graphs

A *regular* graph is a graph in which all the vertices have the same degree. A graph is *planar* if it can be drawn in a plane without its edges crossing. A *face* is a region surrounded by a cycle in a planar embedding of a graph without any path crossing the cycle. A *tree* T_n is a maximal (in terms of size) acyclic graph on n vertices. A *forest* is a disconnected acyclic graph. A *rooted* tree has a distinguished node which is called the *root*. In such a tree, if a node x is one edge away from a node y and the distance of x from the root of the tree is one edge more that the distance of y from the root of the tree, then x is a *child* of y and y is the *parent* of x. A *leaf* is a node with no children. Nodes having the same parent node are called *siblings*. The *height* of a tree T_n, denoted by *Height*(T_n), is

defined as the maximum length of a path from the root of T_n to a leaf of T_n.

A *Cayley tree* is a tree in which each non-leaf vertex has the same degree. A *caterpillar* is a tree in which every vertex is on a central path or only one edge away from the central path (in other words, the removal of its leaves leaves a path). A *lobster* is a tree having the property that the removal of its leaves leaves a caterpillar. A *star* S_n (or n-star graph) of order n, is a tree on n nodes with one node having degree $n-1$ and the other $n-1$ nodes having degree 1. A (n,k)-*banana tree* is a graph obtained by connecting one leaf of each of n copies of a k-star graph with a single root vertex which does not belong to any of the stars. A (n,k)-*firecracker* is a graph obtained by the concatenation of n stars S_k by linking one leaf from each to a path. A *pseudotree*, also called a *unicyclic graph*, is a connected graph with exactly one cycle. A *cactus* graph, sometimes also called a cactus tree, is a connected graph in which any two cycles have no edge in common. Equivalently, it is a connected graph in which any two (simple) cycles have at most one vertex in common. A *Halin* graph is a graph constructed from a plane drawing of a tree having four or more vertices, no vertices of degree two, by connecting all leaves of the tree by a cycle (see Figure 2.1). It is important to mention that leaves are connected in the order they are found in a post order traversal of the ancestor tree of the Halin graph. The first leaf is then connected to the last one (Figure 2.2 illustrates that different orderings of the nodes of a tree may produce different Halin graphs). A *generalized Halin* graph is a Halin graph where we do not have the restriction that vertices do not have degree two. Obviously in the case that only one leaf is present at the ancestor tree of the Halin graph there is no cycle joining its leafs. A *complete* graph on n vertices, denoted by K_n, is a graph in which all n vertices are adjacent to each other. A *bipartite* graph (or *bigraph*) is a

graph whose vertices can be divided into two disjoint sets U and V such that there exists no edge joining two vertices in U or two vertices in V. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles. Obviously if a graph admits a 2-colouring of its vertices it must be bipartite and vice-versa. The *complete bipartite* graph on n and m vertices, denoted by $K_{n,m}$ is the bipartite graph $G = (\{V, U\}, E)$, where V and U are disjoint sets, $|U| = n$, $|V| = m$ and every vertex of V is connected to all vertices of U. A nP_2 stripe graph is the graph consisting of $2n$ vertices and n independent edges.

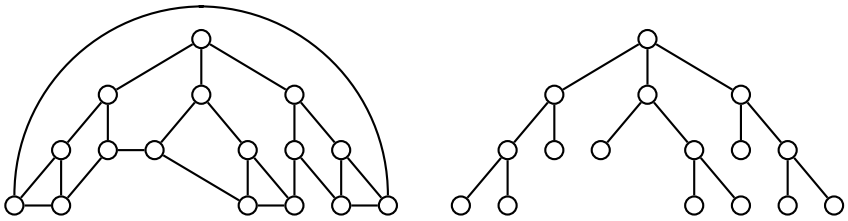


Figure 2.1: A Halin graph with its ancestor tree at its right

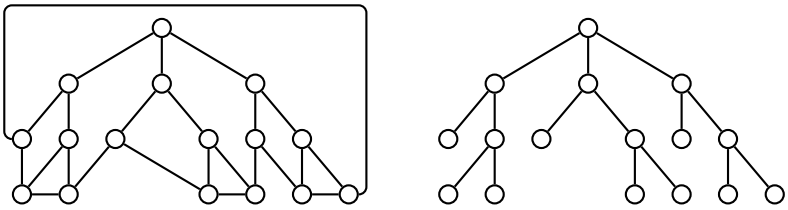


Figure 2.2: A different ordering of the tree of Figure 2.1 produces a different Halin graph

2.2.1 Trees

Almost always in Computer Science, trees are rooted and their nodes are credited with many attributes as shown below.

An *ordered directed graph* G is a pair (N, R), where N is a set of nodes and R is a set of linearly ordered lists of edges such that each element of

R is of the form $((f, g_1), (f, g_2), \ldots, (f, g_m))$, where $f, g_1, g_2, \ldots, g_m \in N$, $m \geq 0$, $g_i \neq f$ for every $1 \leq i \leq m$ and $g_i \neq g_j$ for $i \neq j$. This element would indicate that, for node f, there are m edges leaving f, the first entering node g_1, the second entering node g_2, and so on.

A sequence of nodes $(f_0, f_1, \ldots, f_m)$, $n \geq 1$, is a *directed path* of length m from node f_0 to node f_m if there is an edge which leaves node f_{i-1} and enters node f_i for $1 \leq i \leq n$. A *cycle* is a path $(f_0, f_1, \ldots, f_n)$, where $f_0 = f_n$. An ordered *dag* (dag stands for Directed Acyclic Graph) is an ordered directed graph that has no cycle. A *labelling* of an ordered graph $G = (A, R)$ is a mapping of A into a set of labels drawn from a finite alphabet.

Given a node f, its *out-degree* is the number of distinct pairs $(f, g) \in R$, where $g \in A$. By analogy, the *in-degree* of node f is the number of distinct pairs $(g, f) \in R$, where $g \in A$.

An ordered *tree* t is an ordered dag $t = (N, R)$ with a special node $r \in A$ called the *root* such that

(1) r has in-degree 0,

(2) all other nodes of t have in-degree 1,

(3) there is just one path from the root r to every $f \in N$, where $f \neq r$.

A tree t is *unordered* if no ordering is given on the edge lists of its nodes.

A tree t is *labelled* if every node $f \in N$ is labelled by a symbol $a \in \Sigma$, Σ a finite alphabet.

A tree t is ranked if for every node $f \in N$ the out-degree of f is given.

The number of nodes of a tree t is denoted by $|t|$.

The height of a tree t, denoted by *Height(t)*, is defined as the maximum length of a path from the root of t to a leaf of t.

Let a list of edges of a node f of a tree t be $((f, g_1), (f, g_2), \ldots, (f, g_n))$,

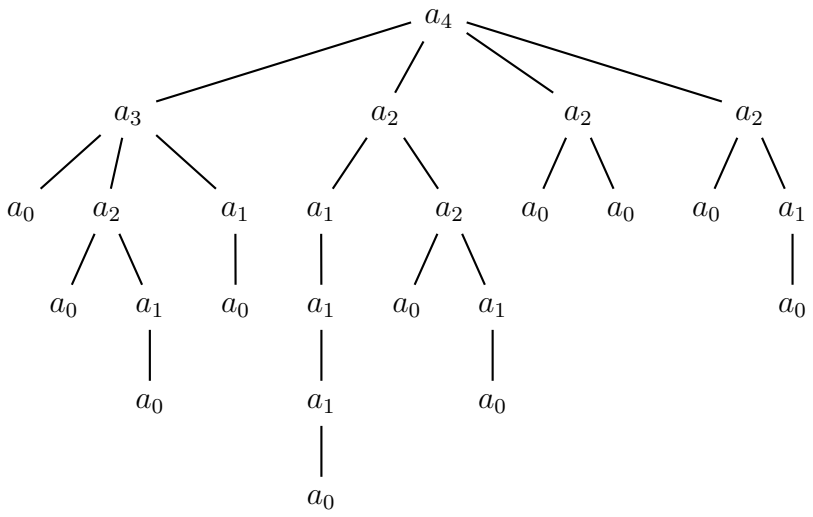


Figure 2.3: Tree t from Example 2.2.1 having postfix notation $post(t) = a_0\,a_0\,a_0\,a_1\,a_2\,a_0\,a_1\,a_3\,a_0\,a_1\,a_1\,a_1\,a_0\,a_0\,a_1\,a_2\,a_2\,a_0\,a_0\,a_2\,a_0\,a_0\,a_1\,a_2\,a_4$

where $f, g_1, g_2, \ldots, g_n \in N$, $n \geq 0$. Then $\{g_1, g_2, \ldots, g_n\}$ are siblings; they are children of f and f is their parent.

The *postfix notation* $post(t)$ of a labelled, ordered, ranked tree t is obtained by applying the following *Step* recursively, beginning at the root of t:

Step: Let the current node be node v. If v is a leaf, list the label of v and halt. If v is an internal node having descendants $v_1, v_2, \ldots, v_{\varphi(v)}$, apply *Step* to $v_1, v_2, \ldots, v_{\varphi(v)}$ in that order and then list the label of v.

Example An illustration of the labelled tree t, having postfix notation $post(t) = a_0\,a_0\,a_0\;\;a_1\,a_2\,a_0\,a_1\,a_3\,a_0\,a_1\,a_1\,a_1\,a_0\,a_0\,a_1\,a_2\,a_2\,a_0\,a_0\,a_2\,a_0\,a_0\,a_1\,a_2\,a_4$ is presented in Fig. 2.3.

A subtree p *matches* a tree T at node n if p is equal to the subtree of t rooted at n.

A *subtree repeat* in a tree T, represented by its postfix notation $x = post(T)$, is a tuple:

$$M_{x,u} = (p; i_1, i_2, \ldots, i_r), r \geq 2$$

where $i_1 < i_2 < \ldots < i_r$ and

$$u = x_{i_1} \ldots x_{i_1+|p|-1} = x_{i_2} \ldots x_{i_2+|p|-1} = \ldots = x_{i_r} \ldots x_{i_r+|p|-1}$$

and $u = post(p)$. If the tuple includes all the occurrences of u in x, then $M_{x,u}$ is said to be complete and is written $M^*_{x,u}$.

2.3 Extremal graph theory

A graph G of order n is an *extremal C_t-free* graph if G has maximum size and girth g at least $t+1$. The set of extremal C_t-free graphs is denoted by $EX(n;t) = EX(n;C_3, C_4, \ldots, C_t)$ and the size of the graph is the extremal number $ex(n;t) = ex(n;C_3, C_4, ..., C_t)$. The largest known lower bound for $ex(n;t)$ is denoted by $ex_l(n;t)$ and the smallest upper bound known by $ex_u(n;t)$.

The Ramsey number $R(m,n)$ is the smallest integer p such that any blue-red colouring of the edges of the complete graph K_p forces the appearance of a blue K_m or a red K_n. The bipartite Ramsey number $R_b(m,n)$ is the smallest integer p such that any blue-red colouring of the edges of the complete bipartite graph $K_{p,p}$ forces the appearance of a blue $K_{m,m}$ or a red $K_{n,n}$. More generally the bipartite Ramsey number $R_b(H,G)$, where H and G are bipartite graphs, is the smallest integer p such that any blue-red colouring of the edges of the complete bipartite graph $K_{p,p}$ forces the appearance of a blue H or a red G.

Zarankiewicz numbers involve bounds on the maximum number of edges in a bipartite graph without a particular subgraph. We denote by $z(p;F)$ the maximum number of edges in a bipartite graph on p vertices without F as a subgraph.

Part III

Words

Chapter 3

Periodicity

The notion of periodicity in words is well studied in many fields like combinatorics on words, pattern matching, data compression and automata theory (see [184, 185]), because it is of paramount importance in several applications, let alone its theoretical aspects.

Algorithms for their identification are an essential part of the research done on words (see [80, 224, 225] for some surveys around the topic). In that direction the KMP algorithm [165] outputs the periods of all prefixes of a word in linear time. Deciding whether or not a word is square-free was first achieved in $O(n \log n)$ time by Rabin [204] using randomization techniques. Later Crochemore [76] described a simpler linear algorithm. Independently, Main and Lorentz [188] have also achieved linear time recognition of square free strings. Leung et al. [180] have worked on online identification of square-free words and they have proposed an $O(h \log^2 h)$ algorithm, where h is the length of the shortest prefix of the word that contains a square. Chen et al. [47] have managed to do the above task using an $O(h \log \beta)$ algorithm, where β is the number of distinct characters in $x[1 \,..\, h]$. Finally, Jansson and Peng [157, 158] gave another $O(h \log h)$ solution.

Identifying all squares of a word is more complicated (applications

of square recognition include DNA tests to determine parentage). An $O(n \log n)$ algorithm, that uses a partitioning of the word into its factors (an idea similar to Hopcroft's partitioning [148]), has been proposed by Crochemore [75]. Using a method based on suffix trees, Apostolico and Preparata [11] suggested an $O(n \log n)$ algorithm for finding all right-maximal repetitions and therefore they were able to identify all squares of a word in that manner. Main and Lorentz [187] proposed another $O(n \log n)$ algorithm which relies on a linear procedure to find all squares of two words when the two words get concatenated and on a variation of the KMP algorithm [165]. A few years later, Kosaraju[171] described a linear time algorithm which finds the shortest square starting from each position of a word. Furthermore, he claims that a modification of his algorithm could output all primitively rooted squares of a word x in $O(n + \Pi(x))$ time, where $\Pi(x)$ is the number of primitively rooted squares in x. Finally, Gusfield [136] has provided a linear time solution by using the suffix tree of the word and marking on it the endpoint of every square that occurs in the word (some early work can be found in [226, 227]).

In 1981, Slisenko [222, 221] proposed a linear solution for the identification of all runs in a word. In 1989, using a special type of factorization called s-factorization, Main [186] proposed a linear time algorithm which finds all leftmost occurrences of distinct runs in a word. More recently, Kolpakov and Kucherov [168, 169] have designed a linear time algorithm for the identification of all runs of a word that uses the suffix tree of the word. This algorithm can be used to identify all squares of the word as well. Crochemore et al. [84] have also designed an $O(n \log n)$ algorithm for the computation of all runs of a string. A modification of Crochemore's algorithm[75] allows the computation of runs [122, 121] in $O(n \log n)$ time.

Palindromes are a special case of periodicity. Identifying the shortest initial palindrome of a word can be done in linear time [189]. Computing the longest palindrome can also be done in linear time [125]. Finally, the problem of identifying all palindromes of a word has been solved in $O(n)$ time [160, 135, 132], using suffix tree techniques. Pan and Lee have also given another algorithm for the same problem [200]. Furthermore, some of the earlier work of the author of [160] on the same problem can be found in [159]. A probabilistic lower bound on palindrome recognition has been discovered in [240].

Apart from algorithmic interest, in the last years a lot of research has been done on bounds on the maximum number of distinct periodicities in a word. These bounds are essential elements of the analysis of some algorithms on strings. The "runs" conjecture, proposed by Kolpakov and Kucherov [168], states that the number of occurrences of maximal repetitions (runs) in a word of length n, is at most n. The first upper bound given was $5n$ [214], which was improved to $3.48n$ [202], to $3.44n$ [215], to $1.6n$ [81], to $1.52n$ for binary words [129] and finally to $1.048n$ [82]. Regarding the lower bound a first estimate of $0.927n$ was given in [123] and improved further to $0.944542n$ [192], to $0.94457567n$ [191] and eventually to $0.944575712n$ [220]. Furthermore, Puglisi and Simpson [203] gave a limit for the expected number of runs in a word in the case that the word length approaches infinity. The exact bounds are still unknown. The maximum number of cubic runs in a word was found to be between $0.5n$ and $0.406n$ [83].

Regarding the maximum number of squares in a word, Fraenkel and Simpson showed that it is at most $2n$ [117], a result proved later in a simpler way by Ilie [149] and improved to $2n - \Theta(\log n)$ [150]. The same number for partial words with one hole was found to be at most $\frac{7n}{2}$ [27]. Kucherov et al. [174] showed that a binary word must contain at least

$0.55080n$ square occurrences. The maximum number of cubes in a word has been shown to be between $\frac{n}{2}$ and $\frac{4n}{5}$ [172]. In a more general scenario, Crochemore et al. [77, 78] proved a $\Theta(n \log n)$ bound on the maximum number of occurrences of primitively rooted kth powers in a word.

As a means to provide more insight into the frequency of appearance of these regularities in words we study the average number of powers and runs of a word of length n using probabilistic methods. We give some bounds for the average number of powers with exponent r in a word, which we then extend for runs and palindromes as well as for some of their abelian varieties (abelian squares and cubes). Finally, as a consequence, we show that a binary word of length n has almost surely $O(n^{\frac{3}{2}})$ abelian squares, where by almost surely we mean that $P(x \text{ has } O(n^{\frac{3}{2}}) \text{ abelian squares})$ tends to 1 as n goes to infinity.

3.1 On the average number of regularities in a word

3.1.1 Properties

As expected, counting powers and runs make use of several types of combinatorial equalities. In this section we quote some combinatorial and arithmetic properties relevant to the problems that we are considering.

The two following lemmas provide us with useful relations for binomial coefficients.

Lemma 3.1.1 *[23]* $\sum_{j=0}^{i} \binom{i}{j}^2 = \binom{2i}{i}$ *for* $i \geq 0$.

Lemma 3.1.2 $\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi n}}$ *as* $n \to \infty$.

A sequence with terms $[a+(k-1)d]r^{k-1}$, where $k \in \mathbb{Z}^+$ and $a, r, d \in \mathbb{R}$, is called arithmetico-geometric and it is the result of the multiplication

of the terms of a geometric sequence with the corresponding terms of an arithmetic sequence.

Lemma 3.1.3 *[210] The sum of the first n terms of an arithmetico-geometric sequence with terms $[a + (k-1)d]r^{k-1}$, where $k \in \mathbb{Z}^+$ and $a, r, d \in \mathbb{R}$, is given by* $S_n = \sum_{k=1}^{n} [a + (k-1)d]\, r^{k-1} = \frac{a}{1-r} - \frac{[a+(n-1)d]r^n}{1-r} + \frac{dr(1-r^{n-1})}{(1-r)^2}$.

We also use the inequalities stated in the next two lemmas whose elementary proofs are similar (see [49]).

Lemma 3.1.4 $\int_1^{n+1} \frac{1}{\sqrt{x}}\,\mathrm{d}x \leq \sum_{i=1}^{n} \frac{1}{\sqrt{i}} \leq \int_0^n \frac{1}{\sqrt{x}}\,\mathrm{d}x$.

Lemma 3.1.5 $\frac{2}{3}n^{\frac{3}{2}} \leq \sum_{i=1}^{n} \sqrt{i} \leq \frac{2}{3}((n+1)^{\frac{3}{2}} - 1)$.

Proof Due to $f(x) = \sqrt{x}$ having a positive first derivative and a negative second derivative the above inequality follows when considering the appropriate areas as shown in Figure 3.1. □

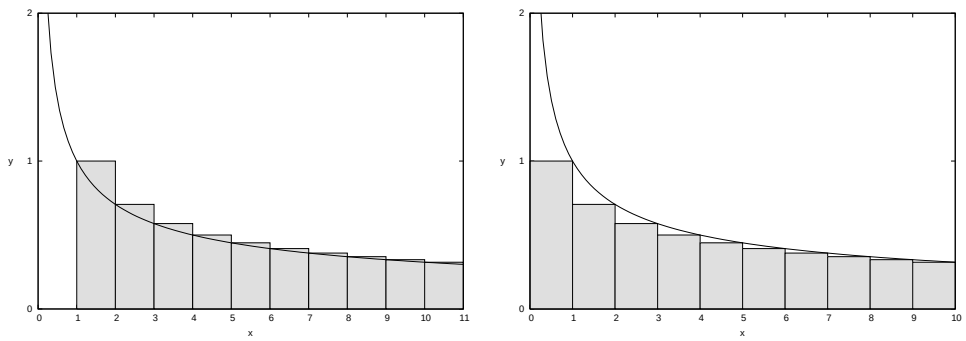


Figure 3.1: Geometrical interpretation of the inequality $\int_0^n \sqrt{x}\,\mathrm{d}x \leq \sum_{i=1}^{n} \sqrt{i} \leq \int_1^{n+1} \sqrt{x}\,\mathrm{d}x$ for $n = 10$

Theorem 3.1.6 (Markov's Inequality) *[96] If x is a random variable and $a \in \mathbb{R}^+$ then $P(|x| \geq a) \leq \frac{E(|x|)}{a}$.*

3.1.2 Powers, runs and palindromes

It seems that regularities appear frequently in a word. In order to comment on that frequency we will study the behaviour of the average number of regularities in Σ^n, the set of words of length n. This average is the same as the expected value of the number of such regularities when we consider a word x, $|x| = n$ with the letters of the word drawn independently from $\Sigma = (a_1, a_2, \ldots, a_\sigma)$ with a constant probability distribution $(\frac{1}{\sigma}, \frac{1}{\sigma}, \ldots, \frac{1}{\sigma})$ on letters.

We show that a word has $\frac{n}{\sigma^{(r-1)}-1} + o(n)$ r-powers on average by using some combinatorial properties and series relations.

Theorem 3.1.7 *On average a word of length n has $\Theta(n)$ r-powers. More precisely this number is* $(n+1)\frac{\sigma^{1-r}(1-\sigma^{(1-r)\lfloor\frac{n}{r}\rfloor})}{1-\sigma^{1-r}} - \frac{r}{\sigma^{r-1}}(\frac{1}{1-\sigma^{1-r}} - \frac{\lfloor\frac{n}{r}\rfloor\sigma^{\lfloor\frac{n}{r}\rfloor(1-r)}}{1-\sigma^{1-r}} + \frac{r(1-\sigma^{(1-r)(\lfloor\frac{n}{r}\rfloor-1)})}{(1-\sigma^{1-r})^2})$.

Proof Let x be a word of length n, with its letters drawn independently from $\{a_1, a_2, \ldots, a_\sigma\}$ with a constant probability distribution $(\frac{1}{\sigma}, \ldots, \frac{1}{\sigma})$.

$$\text{Let } X_{i,j} = \begin{cases} 0, & x[i\,..\,j] \text{ is not an } r\text{-power} \\ 1, & x[i\,..\,j] \text{ is an } r\text{-power} \end{cases}$$

$$\begin{aligned} E(\text{number of } r\text{-powers in } x) &= E(\sum_{i=1}^{n-r+1}\sum_{j=i+1}^{n} X_{i,j}) \\ &= \sum_{i=1}^{n-r+1}\sum_{j=i+1}^{n} E(X_{i,j}) \text{ (linearity of operator)} \\ &= \sum_{i=1}^{n-r+1}\sum_{j=i+1}^{n} P(x[i\,..\,j] \text{ is an } r\text{-power}) \\ &= \sum_{i=1}^{n-r+1}\sum_{j=i+1}^{n} \frac{\sigma^{\frac{j+1-i}{r}}}{\sigma^{j+1-i}}[j+1-i \equiv 0 \bmod r] \end{aligned}$$

$$= \sum_{k=r}^{n} \frac{n+1-k}{\sigma^{\frac{(r-1)k}{r}}}[k \equiv 0 \bmod r]$$

(k is the length of the factor)

$$= \sum_{\ell=1}^{\lfloor \frac{n}{r} \rfloor} \frac{n+1-r\ell}{\sigma^{(r-1)\ell}}$$

$$= (n+1)\sum_{\ell=1}^{\lfloor \frac{n}{r} \rfloor} \frac{1}{\sigma^{(r-1)\ell}} - r\sum_{\ell=1}^{\lfloor \frac{n}{r} \rfloor} \frac{\ell}{\sigma^{(r-1)\ell}}$$

$$= (n+1)\frac{\sigma^{1-r}(1-\sigma^{(1-r)\lfloor \frac{n}{r} \rfloor})}{1-\sigma^{1-r}} - r\sum_{\ell=1}^{\lfloor \frac{n}{r} \rfloor} \frac{\ell}{\sigma^{(r-1)\ell}}$$

$$= (n+1)\frac{\sigma^{1-r}(1-\sigma^{(1-r)\lfloor \frac{n}{r} \rfloor})}{1-\sigma^{1-r}} - \frac{r}{\sigma^{r-1}}\Big(\frac{1}{1-\sigma^{1-r}}$$

$$- \frac{\lfloor \frac{n}{r} \rfloor \sigma^{\lfloor \frac{n}{r} \rfloor (1-r)}}{1-\sigma^{1-r}} + \frac{r(1-\sigma^{(1-r)(\lfloor \frac{n}{r} \rfloor - 1)})}{(1-\sigma^{1-r})^2}\Big) \text{ (Lemma 3.1.3)} \qquad \square$$

The theorems below give some bounds on the average number of runs in a word.

Theorem 3.1.8 *On average a word of length n has $O(n)$ runs. More precisely this number is at most $\frac{n}{\sigma} + o(n)$.*

Proof Let y=number of runs in x. By proceeding as in the proof of Theorem 3.1.7 we get:

$$E(y) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} P(x[i\,..\,j] \text{ is a run})$$

$$= \sum_{j=2}^{n-1} P(x[1\,..\,j] \text{ is a run}) + P(x[1\,..\,n] \text{ is a run})$$

$$+ \sum_{i=2}^{n-2}\sum_{j=i+1}^{n-1} P(x[i\,..\,j] \text{ is a run}) + \sum_{i=2}^{n-1} P(x[i\,..\,n] \text{ is a run})$$

$$= \sum_{j=2}^{n-1} \sum_{\substack{k=1 \\ x[1\,..\,k] \text{ is primitive}}}^{\lfloor \frac{j}{2} \rfloor} \frac{\sigma^k(\sigma-1)}{\sigma^{j+1}} + \sum_{i=2}^{n-2}\sum_{j=i+1}^{n-1} \sum_{\substack{k=1 \\ x[i\,..\,i+k-1] \text{ is primitive}}}^{\lfloor \frac{j-i+1}{2} \rfloor} \frac{\sigma^k(\sigma-1)^2}{\sigma^{j-i+3}}$$

$$+ \sum_{\substack{k=1 \\ x[1\,..\,k] \text{ is primitive}}}^{\lfloor \frac{n}{2} \rfloor} \frac{\sigma^k}{\sigma^n} + \sum_{i=2}^{n-1} \sum_{\substack{k=1 \\ x[i\,..\,i+k-1] \text{ is primitive}}}^{\lfloor \frac{n-i+1}{2} \rfloor} \frac{\sigma^k(\sigma-1)}{\sigma^{n-i+2}}$$

$$\leq \sum_{j=2}^{n-1} \sum_{k=1}^{\lfloor \frac{j}{2} \rfloor} \frac{\sigma^k(\sigma-1)}{\sigma^{j+1}} + \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\sigma^k}{\sigma^n} + \sum_{i=2}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=1}^{\lfloor \frac{j-i+1}{2} \rfloor} \frac{\sigma^k(\sigma-1)^2}{\sigma^{j-i+3}}$$

$$+ \sum_{i=2}^{n-1} \sum_{k=1}^{\lfloor \frac{n-i+1}{2} \rfloor} \frac{\sigma^k(\sigma-1)}{\sigma^{n-i+2}}$$

$$= \sum_{i=2}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=1}^{\lfloor \frac{j-i+1}{2} \rfloor} \frac{\sigma^k(\sigma-1)^2}{\sigma^{j-i+3}} + o(n)$$

$$= \sum_{\ell=2}^{n-2} \sum_{k=1}^{\lfloor \frac{\ell}{2} \rfloor} (n-1-\ell) \frac{\sigma^k(\sigma-1)^2}{\sigma^{\ell+2}} + o(n) \text{ (ℓ is the length of the factor)}$$

$$= \frac{n(\sigma-1)^2}{\sigma^2} \sum_{\ell=2}^{n-2} \sum_{k=1}^{\lfloor \frac{\ell}{2} \rfloor} \frac{\sigma^k}{\sigma^\ell} + o(n) \text{ (Lemma 4.2.3)}$$

$$= \frac{n(\sigma-1)}{\sigma} \sum_{\ell=2}^{n-2} \frac{\sigma^{\lfloor \frac{\ell}{2} \rfloor} - 1}{\sigma^\ell} + o(n)$$

$$= \frac{n(\sigma-1)}{\sigma} \sum_{\ell=1}^{n-2} (\sigma^{-\lceil \frac{\ell}{2} \rceil} - \sigma^{-\ell}) + o(n)$$

$$= \frac{n(\sigma-1)}{\sigma} (2 \sum_{i=1}^{\lceil \frac{n-2}{2} \rceil} \sigma^{-i} - [n \equiv 1 \bmod 2] \sigma^{-\frac{n-1}{2}}$$

$$- \frac{\sigma^{-1}(\sigma^{-(n-2)} - 1)}{\sigma^{-1} - 1}) + o(n)$$

$$= \frac{n(\sigma-1)}{\sigma} (2 \frac{\sigma^{-1}}{1-\sigma^{-1}} - \frac{1}{\sigma-1}) + o(n) = \frac{n}{\sigma} + o(n)$$

Lemma 4.2.3 suggests that the above series are bounded by constant terms, thus proving the theorem. □

Puglisi and Simon [203] using Möbius functions and several combinatorial properties, instead of probabilistic tools, showed that the expected number of runs per unit length in a word on alphabet of size σ tends to $\frac{\sigma-1}{\sigma} \sum_{i=1}^{\infty} \frac{\mu(i)}{\sigma^{2i-1}-1}$ as the word length approaches infinity. Here μ is the Möbius function ($\mu : \mathbb{Z}^+ \to \{-1, 0, 1\}$), where $\mu(1) = 1$ and if $n \geq 2$

and the prime factorization of n is $\prod_{i=1}^{t} p_i^{a_i}$ then $\mu(n) = 0$ if $a_i > 1$ for any i and $(-1)^t$ otherwise. Therefore the expression $\frac{\sigma-1}{\sigma}\sum_{i=1}^{\infty}\frac{\mu(i)}{\sigma^{2i-1}-1}$ is strictly less than $\frac{1}{\sigma-1}+\frac{\sigma-1}{\sigma}\sum_{i=1}^{\infty}\frac{1}{\sigma^i}=\frac{1}{\sigma}+\frac{1}{\sigma-1}$, which verifies our result.

Similarly we get some bounds on the average number of palindromes in a word.

Theorem 3.1.9 *On average a word of length n has $\Theta(n)$ palindromes. More precisely this number is* $\frac{\sigma+1}{\sigma-1}\cdot n+(\frac{2}{1-\sigma}-\frac{1}{2}\cdot[n \bmod 2=0])\cdot n\sigma^{-\lfloor\frac{n}{2}\rfloor}+\frac{2}{\sigma-1}\cdot\lfloor\frac{n}{2}\rfloor\sigma^{-\lfloor\frac{n}{2}\rfloor}-\frac{2}{(1-\sigma^{-1})^2}-([n \bmod 2=0]+\frac{4}{\sigma-1}+\frac{2}{\sigma(1-\sigma^{-1})^2})\sigma^{-\lfloor\frac{n}{2}\rfloor}$

Proof Let y =number of palindromes in x. By proceeding as in the proof of Theorem 3.1.7 we get:

$$
\begin{aligned}
E(y) &= n+\sum_{i=1}^{n}\sum_{j=i+1}^{n} P(x[i\,..\,j] \text{ is a palindrome}) = n+\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{\sigma^{\lceil\frac{j+1-i}{2}\rceil}}{\sigma^{j+1-i}}\\
&= n+\sum_{k=2}^{n}\frac{n+1-k}{\sigma^{\lfloor\frac{k}{2}\rfloor}} \ (k \text{ is the length of the factor})\\
&= n+(n+1)\sum_{k=2}^{n}\frac{1}{\sigma^{\lfloor\frac{k}{2}\rfloor}}-\sum_{k=2}^{n}\frac{k}{\sigma^{\lfloor\frac{k}{2}\rfloor}}\\
&= n+(n+1)\left(2\sum_{\ell=1}^{\lfloor\frac{n}{2}\rfloor}\frac{1}{\sigma^{\ell}}-[n \bmod 2=0]\frac{1}{\sigma^{\frac{n}{2}}}\right)-2\sum_{\ell=1}^{\lfloor\frac{n}{2}\rfloor}\frac{\ell}{\sigma^{\ell}}\\
&+[n \bmod 2=0]\frac{n}{2\cdot\sigma^{\frac{n}{2}}}\\
&= n+(n+1)\left(2\cdot\frac{1-\sigma^{-\lfloor\frac{n}{2}\rfloor}}{\sigma-1}-[n \bmod 2=0]\frac{1}{\sigma^{\frac{n}{2}}}\right)-\frac{2}{\sigma}\cdot\frac{1}{1-\sigma^{-1}}\\
&+\frac{2}{\sigma}\cdot\frac{[1+(\lfloor\frac{n}{2}\rfloor-1)]\sigma^{-\lfloor\frac{n}{2}\rfloor}}{1-\sigma^{-1}}-\frac{2}{\sigma}\cdot\frac{\sigma^{-1}(1-\sigma^{1-\lfloor\frac{n}{2}\rfloor})}{(1-\sigma^{-1})^2}\\
&+[n \bmod 2=0]\frac{n}{2\cdot\sigma^{\frac{n}{2}}} \ (\text{Lemma 3.1.3})\\
&= \frac{\sigma+1}{\sigma-1}\cdot n+(\frac{2}{1-\sigma}-\frac{1}{2}\cdot[n \bmod 2=0])\cdot n\sigma^{-\lfloor\frac{n}{2}\rfloor}\\
&+\frac{2}{\sigma-1}\cdot\lfloor\frac{n}{2}\rfloor\sigma^{-\lfloor\frac{n}{2}\rfloor}-\frac{2}{(1-\sigma^{-1})^2}-([n \bmod 2=0]+\frac{4}{\sigma-1}\\
&+\frac{2}{\sigma(1-\sigma^{-1})^2})\sigma^{-\lfloor\frac{n}{2}\rfloor}
\end{aligned}
$$

□

3.1.3 Abelian powers

Recall that an abelian square is a word of the form yz for two nonempty words y and z having the same letter composition (same number of occurrences of each letter). The definition extends to other powers.

Theorem 3.1.10 *A binary word of length n has $\Theta(n^{\frac{3}{2}})$ abelian squares on average. More precisely this number is $\frac{2\sqrt{2}}{3\sqrt{\pi}} \cdot n^{\frac{3}{2}} + o(n^{\frac{3}{2}})$.*

Proof Let y be the number of abelian squares in x. By proceeding as in the proof of Theorem 3.1.7 we get:

$$
\begin{aligned}
E(y) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} P(x[i\,..\,j] \text{ is an abelian square}) \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{\sum_{\ell=0}^{j+1-i} \binom{\frac{j+1-i}{2}}{\ell} \binom{\frac{j+1-i}{2}}{\ell}}{2^{j+1-i}} [i \not\equiv j \bmod 2] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{\binom{j+1-i}{\frac{j+1-i}{2}}}{2^{j+1-i}} [i \not\equiv j \bmod 2] \text{ (Lemma 3.1.1)} \\
&= \sum_{k=2}^{n} (n+1-k) \frac{\binom{k}{\frac{k}{2}}}{2^k} [k \equiv 0 \bmod 2] \text{ (k is the length of the factor)} \\
&= \sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} (n+1-2\ell) \frac{\binom{2\ell}{\ell}}{4^\ell} \\
&= (n+1) \sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\binom{2\ell}{\ell}}{4^\ell} - 2 \sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\ell}{4^\ell} \binom{2\ell}{\ell} \\
&= (n+1) \sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\sqrt{\pi \ell}} - 2 \sum_{\ell=1}^{\lfloor \frac{n}{2} \rfloor} \sqrt{\frac{\ell}{\pi}} + o(n^{\frac{3}{2}}) \text{ (Lemma 3.1.2)} \\
&= \frac{2\sqrt{2}}{3\sqrt{\pi}} \cdot n^{\frac{3}{2}} + o(n^{\frac{3}{2}}) \text{ (Lemmas 3.1.4 and 3.1.5)}
\end{aligned}
$$

□

Before proceeding with the following theorem, let us recall that Franel numbers are defined as:

$$\mathrm{Fr}_n = \sum_{i=0}^{n} \binom{n}{i}^3 \qquad \text{where } i \in \mathbb{Z}^+$$

Theorem 3.1.11 *The average number of abelian cubes in a binary word of length n can be expressed as a sum involving Franel numbers, namely* $\sum_{m=1}^{\lfloor \frac{n}{3} \rfloor}(n+1-3m)\frac{\mathrm{Fr}_m}{2^{3m}}$.

Proof Let y be the number of abelian cubes in x. By proceeding as in the proof of Theorem 3.1.7 we get:

$$\begin{aligned}
E(y) &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} P(x[i\,..\,j] \text{ is an abelian cube}) \\
&= \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{\sum_{\ell=0}^{\frac{j+1-i}{3}} \binom{\frac{j+1-1}{3}}{\ell}^3}{2^{j+1-i}} [j+1-i \equiv 0 \bmod 3] \\
&= \sum_{k=3}^{n}(n+1-k)\frac{\sum_{\ell=0}^{\frac{k}{3}} \binom{\frac{k}{3}}{\ell}^3}{2^k} [k \equiv 0 \bmod 3] \text{ (k is the length of the} \\
&\quad \text{factor)} \\
&= \sum_{m=1}^{\lfloor \frac{n}{3} \rfloor}(n+1-3m)\frac{\sum_{\ell=0}^{m} \binom{m}{\ell}^3}{2^{3m}} \\
&= \sum_{m=1}^{\lfloor \frac{n}{3} \rfloor}(n+1-3m)\frac{\mathrm{Fr}_m}{2^{3m}}
\end{aligned}$$

□

3.1.4 Consequences

By applying Markov's inequality and using Theorem 3.1.10, we observe that a large word almost surely has $O(n^{\frac{3}{2}})$ abelian squares. More formally:

Theorem 3.1.12 *In the case that x is a binary word, the probability $P(x$ has $O(n^{\frac{3}{2}})$ abelian squares$)$ tends to 1 as n goes to infinity.*

Proof Let y be the number of abelian squares of x. Applying Markov's

inequality with $a = bn^{\frac{3}{2}+\epsilon}$, where $b \in \mathbb{R}^+$ and $\epsilon \in (0, \frac{1}{2}]$, gives:

$$P(|y| \geq bn^{\frac{3}{2}+\epsilon}) \leq \frac{E(|y|)}{bn^{\frac{3}{2}+\epsilon}} = \frac{\frac{2\sqrt{2}}{3\sqrt{\pi}}n^{\frac{3}{2}} + o(n^{\frac{3}{2}})}{bn^{\frac{3}{2}+\epsilon}}$$
$$< \frac{n^{\frac{3}{2}}}{bn^{\frac{3}{2}+\epsilon}} = \frac{1}{bn^{\epsilon}} \to 0 \text{ as } n \to +\infty$$

The last statement shows that $P(x \text{ has } \omega(n^{\frac{3}{2}}) \text{ abelian squares}) \to 0$ as $n \to +\infty$, deducing the statement in the beginning of the theorem. □

It is worth noting that one can get similar results for other regularities using the same techniques.

We summarize our results in the following table:

Regularity	**Average number of occurrences in a word of Σ^n**
r-powers	$\frac{n}{\sigma^{(r-1)}-1} + o(n)$
Runs	at most $\frac{n}{\sigma} + o(n)$
Palindromes	$(1 + \frac{2}{\sigma-1})n + o(n)$
Abelian squares	$\frac{2\sqrt{2}}{3\sqrt{\pi}}n^{\frac{3}{2}} + o(n^{\frac{3}{2}})$ (holds for binary words)

It is important to mention that for the derivation of our results similar methods are used. The results are numerically derived with no structural properties of strings used. Therefore they could be summarised in the form of a multi theorem.

Chapter 4

Quasiperiodicity

The concept of quasiperiodicity is a generalization of the notion of periodicity. In a periodic repetition the occurrences of the single periods do not overlap. In contrast, the quasiperiods of a word may overlap. For some surveys on the topic see [7, 8].

There are many algorithms for detecting quasiperiodicities. An $O(n)$ superprimitivity testing was proposed by Apostolico et al. in 1991 [10]. Independently, Breslauer [33] gave another linear time algorithm for the same problem. His algorithm is not only online but it computes also the cover array of the given string (containing the length of the shortest cover of each prefix of the given string). The same author gave also an $O(\log(\log n))$ parallel algorithm that tests if a word is superprimitive [34] and uses $\frac{n \log n}{\log(\log n)}$ processors. Apostolico and Ehrenfeucht [9] managed to detect maximal quasiperiodic subwords in $O(n \log^2 n)$ time. In 1999, Iliopoulos and Mouchard [155] and later Brodal and Petersen [36] proposed $O(n \log n)$ algorithms for identifying all maximal quasiperiodicities in a string. Groult and Richomme [133] proved the optimality of the above algorithms.

A fundamental problem is to find all the covers of a string y of length n. Linear-time algorithms were given by Moore and Smyth in [196, 197],

and by Li and Smyth in [181]. An $O(\log(\log n))$-time work-optimal parallel algorithm was given later by Iliopoulos and Park in [153]. A close and also fundamental problem is that of computing the shortest (resp. longest) cover of every prefix of a string. This gives rise to the so-called *minimal cover* (resp. *maximal cover*) *array*. An integer array C is the minimal cover (resp. maximal cover) array of y, if $\mathsf{C}[i]$ is the minimal (resp. maximal) length of covers of $y[0\,..\,i]$. Apostolico and Breslauer [7, 33] gave an online linear-time algorithm for computing the minimal cover array of a string, using the algorithm by Knuth, Morris and Pratt [165] for computing the period of every prefix of a string in linear time. In addition, Li and Smyth in [181], provided an algorithm, having the same characteristics, for computing the maximal cover array of a given string; this algorithm gives also all the covers for every prefix of the string.

Seeds were first defined and studied by Iliopoulos, Moore and Park in [154]. An $O(n \log n)$ algorithm for the corresponding problem on seeds was given in [154] and later a linear time algorithm has been proposed by Kociumaka et al. [166]. Recently Christou et al. gave a linear-time algorithm for computing the minimal/maximal left-seed array of y [61] (i.e. the length of the shortest left seed of each prefix of the string), an $O(n \log n)$-time algorithm for computing the minimal right-seed array of y (i.e. the length of the shortest right seed of each prefix of the string), and a linear-time solution for computing the maximal right-seed array of y [57]. A quadratic time algorithm for the computation of the seed array has been proposed in [61] while Christodoulakis et al. gave some polynomial time algorithms for the approximate seed problem [54].

Much research has concentrated around algorithms for the computation of quasiperiodicities in strings while not much is known about bounds on their number of occurrences in words. Some research has been done in that direction, mainly related to Fibonacci words, e.g. identification

of all covers of a circular Fibonacci word [152] and identification of all maximal quasiperiodicities in Fibonacci words [133]. Furthermore, the infinite Thue-Morse word was the first example of an infinite overlap-free word [6, 232, 233], proven to be the only infinite binary word having that property [219]. Additionally, it was shown that there is an infinite number of overlap-free infinite partial words with one hole, but none in infinite words with more than one hole [139].

Regarding seeds, we show a linear-time algorithm for computing the minimal left-seed array of x [64, 59, 65], a linear-time solution for computing the maximal left-seed array of x[64, 59, 65], an $O(n \log n)$ time algorithm for computing the minimal right-seed array of x[58, 59] and a linear-time solution for computing the maximal right-seed array of x[58, 59]. Our algorithms have used other algorithms for computing the period array [165] and the cover array of a string ([7, 33]). It is important to note that all of the proposed algorithms use linear auxiliary space. We also study the frequency of appearance of seeds in words. Using combinatorial properties of words we prove that a word has $O(n)$ seeds on average [50]. Furthermore we show that the maximum number of distinct seeds in a word is between $\frac{1}{6}n^2 + o(n^2)$ and $\frac{1}{4}n^2 + o(n^2)$ and we reveal some properties for the structure of an extremal word for the last case [50]. It is important to note that we restrict seeds to be factors of the given word. Furthermore, we study the overlapping factors of a word as a means to provide more insight into quasiperiodic structures of words. We propose a linear time algorithm for the identification of all overlapping factors of a word and we provide some bounds on the maximum number of distinct overlapping factors in a word [52].

4.1 Left-seed and Right-seed arrays

Before showing the algorithms for the computation of left-seed arrays and right-seed arrays we need to introduce some more definitions.

The *border array* B of x is the array of integers $\mathsf{B}[1\,..\,n]$ for which $\mathsf{B}[i]$, $1 \leq i \leq n$, stores the length of the border of the prefix $x[1\,..\,i]$.

The *period array* P of x is the array of integers $\mathsf{P}[1\,..\,n]$ for which $\mathsf{P}[i]$, $1 \leq i \leq n$, stores the length of the period of the prefix $x[1\,..\,i]$.

The *minimal cover array* C of x is the array of integers $\mathsf{C}[1\,..\,n]$ for which $\mathsf{C}[i]$, $1 \leq i \leq n$, stores the length of the shortest cover of the prefix $x[1\,..\,i]$. The *maximal cover array* C^{M} of x is the array of integers $\mathsf{C}^{\mathrm{M}}[1\,..\,n]$ for which $\mathsf{C}^{\mathrm{M}}[i]$, $1 \leq i \leq n$, stores the length of the longest cover of the prefix $x[1\,..\,i]$ which is smaller than x (0 if none).

The *minimal left-seed array* LS of x is the array of integers $\mathsf{LS}[1\,..\,n]$ for which $\mathsf{LS}[i]$, $1 \leq i \leq n$, stores the length of the minimal left seed of the prefix $x[1\,..\,i]$. The *maximal left seed* of x, denoted by $\mathrm{Mls}(x)$, is the longest prefix of x that is a cover of a superstring of x. The *maximal left-seed array* LS^{M} of x is the array of integers $\mathsf{LS}^{\mathrm{M}}[1\,..\,n]$ for which $\mathsf{LS}^{\mathrm{M}}[i]$, $1 \leq i \leq n$, stores the length of the maximal left seed of the prefix $x[1\,..\,i]$, which is smaller than i (0 if none). The *minimal right seed* of x, denoted by $\mathrm{mrs}(x)$, is the shortest suffix of x that is a cover of a superstring of y. The *minimal right-seed array* RS of x is the array of integers $\mathsf{RS}[1\,..\,n]$ for which $\mathsf{RS}[i]$, $1 \leq i \leq n$, stores the length of the minimal right seed of the prefix $x[1\,..\,i]$. The *maximal right seed* of x, denoted by $\mathrm{Mrs}(x)$, is the longest suffix of x that is a cover of a superstring of y. The *maximal right-seed array* RS^{M} of x is the array of integers $\mathsf{RS}^{\mathrm{M}}[1\,..\,n]$ for which $\mathsf{RS}^{\mathrm{M}}[i]$, $1 \leq i \leq n$, stores the length of the maximal left seed of the prefix $x[1\,..\,i]$, which is smaller than i (0 if none).

The following example shows B, P, C, C^{M}, LS, LS^{M}, RS, and RS^{M} for the string $x = \texttt{abaababaabaabab}$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x[i]$	a	b	a	a	b	a	b	a	a	b	a	a	b	a	b
$\mathsf{B}[i]$	0	0	1	1	2	3	2	3	4	5	6	4	5	6	7
$\mathsf{P}[i]$	1	2	2	3	3	3	5	5	5	5	5	8	8	8	8
$\mathsf{C}[i]$	1	2	3	4	5	3	7	3	9	5	3	12	5	3	15
$\mathsf{C}^{\mathrm{M}}[i]$	0	0	0	0	0	3	0	3	0	5	6	0	5	6	0
$\mathsf{LS}[i]$	1	2	2	3	3	3	3	3	3	3	3	3	3	3	13
$\mathsf{LS}^{\mathrm{M}}[i]$	0	0	2	3	4	5	6	7	8	9	10	11	12	13	14
$\mathsf{RS}[i]$	1	2	2	3	3	3	5	3	5	5	3	8	5	3	8
$\mathsf{RS}^{\mathrm{M}}[i]$	0	0	2	3	4	5	6	7	8	9	10	11	12	13	14

4.1.1 Properties

In this subsection, we prove some useful properties of left seeds, of right seeds, and of covers of a given string x.

The following lemma gives the condition for a string to be a left seed of another string.

Lemma 4.1.1 ([64]) *A string z is a left seed of x iff it covers a prefix of x whose length is at least the period of x.*

Proof Direct: Suppose a string z covers a prefix of x, say uv, larger or equal to $\mathrm{Per}(x)$, where $|u| = \mathrm{Per}(x)$ and v is a possibly non empty string. Let k the smallest integer such that x is a prefix of u^k. Then z is a cover of $u^k v = xwv$, for some string w, possibly empty. Therefore z is a left seed of x.

Reverse: Let z be a left seed of x.

- if $|z| \leq \mathrm{Border}(x)$. Then a suffix v of z, possibly empty, is a prefix of the border (consider the left seed that covers $x[\mathrm{Per}(x)]$). Then z is a cover of uv, where u is the period of x.

- if $|z| > \text{Border}(x)$. Let z not a cover of a prefix of x larger or equal to $\text{Per}(x)$. Let v a border of x such that $|v| = \text{Border}(x)$. Then v is a factor of z, such that $z = uvw$, where u and w are non empty words (consider the left seed that covers $x[\text{Per}(x)]$). This gives uv a longest border for x, which is a contradiction.

□

Similarly, the following lemma gives the condition for a string to be a right seed of another string.

Lemma 4.1.2 ([58]) *A string z is a right seed of x iff it covers a suffix of x whose length is at least the period of x.*

Proof Direct consequence of Lemma 4.1.1. □

Lemma 4.1.3 ([196]) *Let u be a proper cover of x, in our case some prefix $x[1\,..\,i]$ of x, and let $z \neq u$ be a substring of x such that $|z| \leq |u|$. Then z is a cover of x if and only if z is a cover of u.*

Proof Clearly if z is a cover of u and u a cover of x then z is a cover of x. Suppose now that both z and u cover x. Then z is a border of x and hence of u ($|z| \leq |u|$); thus z must also be a cover of u. □

The following lemma shows that an aperiodic string has no left seeds other than itself.

Lemma 4.1.4 ([64]) *Let x be a string of length n and $\text{Per}(x) = n$, then the minimal left seed of x is x.*

Proof By definition of the minimal left seed: $|\text{mls}(x)| \leq n$. Let $|\text{mls}(x)| < n$. Then, in order to cover x, a non-empty prefix of $\text{mls}(x)$, say w, is a suffix of x (consider the left seed that covers $x[n]$). Then $n - |w|$ gives a shorter period for x, which is a contradiction. □

The following lemma shows that values in the left seed array of a string are increasing.

Lemma 4.1.5 ([64]) *Let* $LS[i] = |\text{mls}(x[1\,..\,i])|$, *for all* $1 \leq i \leq n$, *then* $LS[i] \leq LS[i+1]$, *for all* $1 \leq i \leq n-1$.

Proof Let $\mathsf{LS}[i] > \mathsf{LS}[i+1]$. By definition of the minimal left seed: $\mathsf{LS}[i]$ covers some superstring $x[1\,..\,i]u$, where u is a possibly empty string, and $\mathsf{LS}[i+1]$ covers some superstring $x[1\,..\,i+1]v$, where v is a possibly empty string. In other words $\mathsf{LS}[i+1]$ covers $x[1\,..\,i]x[i+1]v$. Therefore, by definition of the minimal left seed, $\mathsf{LS}[i+1]$ is a minimal left seed of $x[1\,..\,i]$. But then we get a shorter left seed for $x[1\,..\,i]$, which is a contradiction. □

The following lemma shows the maximal left seed of a string depending whether the string is aperiodic or not.

Lemma 4.1.6 ([64]) *Let* x *be a string of length* n *and* $\text{Per}(x) = k$, *then*

- *if* $k = n$, *then there is no maximal left seed for* x
- *if* $k < n$, *then the maximal left seed of* x *is* $x[1\,..\,n-2]$

Proof

- if $k = n$, then, by definition of the maximal left seed, it holds that $|\text{mls}(x)| < n$. Let $x[1\,..\,j]$ be the maximal left seed of x, with $1 \leq j \leq n$. Then, in order to cover x, a non-empty prefix of $x[1\,..\,j]$, say w, is a suffix of x (consider the maximal left seed that covers $x[n]$). Then $n - |w|$ gives a shorter period for x, which is a contradiction.
- if $k < n$, then the maximal left seed of x is $x[1..n-1]$, as it is a cover of the superstring $xx[\text{Border}(x)+1\,..\,n-1]$ of x and it has the maximum length allowed, which is $n-1$.

□

The following lemma shows the maximal right seed of a string depending whether the string is aperiodic or not.

Lemma 4.1.7 ([58]) *Let x be a string of length n and* $\mathrm{Per}(x) = k$*, then*

- *if $k = n$, then there is no maximal right seed for x.*
- *if $k < n$, then the maximal right seed of x is $x[1..n-1]$.*

Proof Similar to the proof of Lemma 4.1.6. □

4.1.2 Computing the minimal and the maximal left-seed array

In this subsection, we describe our algorithms for computing the minimal and maximal left-seed array of a string. In order to find the minimal left-seed array of a string, we use the algorithm for computing the minimal cover array by Apostolico and Breslauer [7, 33] in linear time, and Lemma 4.1.1, which gives the necessary and sufficient condition for a prefix of a string to be a left seed of that string. Finding the maximal left-seed array reduces to detecting border-free prefixes of the given string.

Algorithm R-ARRAY computes an array R, which stores the length of the longest prefix of x that is covered by $x[1\,..\,i]$, 0 if none. It takes as input the minimal cover array C.

ALGORITHM R-ARRAY(C, n)
1: **for** $i \leftarrow 1$ **to** n **do** R[i] $\leftarrow 0$;
2: **for** $i \leftarrow 1$ **to** n **do** R[C[i]] $\leftarrow i$;
3: **return** R[$1\,..\,n$];

Algorithm MINIMAL-LEFT-SEED-ARRAY computes an array LS, which stores the length of the minimal left seed of $x[1\,..\,i]$. It takes as input the array R and the period array P.

ALGORITHM MINIMAL-LEFT-SEED-ARRAY(x, n, P,R)
1: **for** $i \leftarrow 1$ **to** n **do**
2: **if** $\mathsf{P}[i] \leftarrow i$ **do** $\mathsf{LS}[i] \leftarrow i$;
3: **else**
4: $ls \leftarrow \mathsf{LS}[i-1]$;
5: **while** $\mathsf{R}[ls] < \mathsf{P}[i]$ **do** $ls \leftarrow ls + 1$;
6: $\mathsf{LS}[i] \leftarrow ls$;
7: **return** $\mathsf{LS}[1\,..\,n]$;

Then, we proceed to compute the array LS. There are two cases:

- if $\mathsf{P}[i] = i$, then by Lemma 4.1.4, $\mathsf{LS}[i] = i$
- if $\mathsf{P}[i] < i$, then by Lemma 4.1.5, we must look for $\mathsf{LS}[i]$ in the set $\{\mathsf{LS}[i-1], ..., n\}$

Lemma 4.1.1 gives the necessary and sufficient condition ($\mathsf{R}[ls] \geq \mathsf{P}[i]$), i.e. the length of the longest prefix of x that is covered by $x[1\,..\,ls]$ to be greater than or equal to the period of $x[1\,..\,i]$, where ls is the candidate length of the minimal left seed of $x[1\,..\,i]$.

Before proceeding with the following theorem we must introduce the indicator function:

$$I : X \to \{0, 1\}$$

such that as $I(x \in A) = I_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A, \end{cases}$
where X is a set and A a subset of X.

Theorem 4.1.8 *Computing the minimal left-seed array of a string x of length n can be done in linear time.*

Proof Arrays C, P and R can be computed in linear time as of [7, 33, 79, 165]. Then, we only have to prove that the process to compute array LS runs in linear time, as well. Let c_i be the number of comparisons done

in the second loop at step i. Then clearly:

$$\begin{aligned}
\mathrm{T} &= \text{Time needed} \\
&= 2n + 2 + \sum_{i=1}^{n} (1 + I(\mathsf{P}[i] = i) + I(\mathsf{P}[i] \neq i) * (2 + 2c_i - 1)) \\
&= 2n + 2 + \sum_{i=1}^{n} (2 + I(\mathsf{P}[i] \neq i) * 2c_i) \\
&= 2n + 2 + 2n + \sum_{i=1}^{n} 2c_i I(\mathsf{P}[i] \neq i)
\end{aligned}$$

Therefore

$$\mathrm{T} \leq 4n + 2 + 2\sum_{i=1}^{n} c_i \tag{4.1}$$

Let s_i be the set of values checked in the second loop at step i. Clearly $s_i \cap s_{i+1} = 1$ given by the 1st value to be checked in s_{i+1}, which is the same as the last value in s_i. Then

$$c_i + c_{i+1} = |s_i| + |s_{i+1}| = |s_i \cup s_{i+1}| + |s_i \cap s_{i+1}| \tag{4.2}$$

$$c_i + c_{i+1} = |s_i \cup s_{i+1}| + 1 \tag{4.3}$$

By Equations 4.1 and 4.3:

$$T \leq 4n + 2 + 2(n - 1 + |\bigcup_{i=1}^{n} s_i|) \tag{4.4}$$

As $\bigcup_{i=1}^{n} s_i$ is at most $\{1, ..., n\}$ then $|\bigcup_{n}^{i=1} s_i| \leq n$, and so $T \leq 8n$. Therefore $T = O(n)$. □

For the computation of the maximal left-seed array, we can use Lemma 4.1.6 to obtain the following two cases.

- if $\mathsf{P}[i] = i$, then $\mathsf{LS}^{\mathrm{M}}[i] = 0$
- if $\mathsf{P}[i] < i$, then $\mathsf{LS}^{\mathrm{M}}[i] = i - 1$

Hence, we obtain the following result.

Theorem 4.1.9 *Computing the maximal left-seed array of a string x of length n can be done in linear time.*

4.1.3 Computing the minimal and the maximal right-seed array

In this subsection, we describe our algorithms for calculating the minimal and the maximal right-seed array of a string. In order to identify the minimal right-seed array of a string, we use a variant of the partitioning algorithm in [75, 154] to find the sets of ending positions of all the occurrences of each factor in the string. We are then able to find which suffix of each prefix of the string is covered by that factor, and check for right seeds using Lemma 4.1.2. Computing the maximal right-seed array of a string reduces to detecting border-free prefixes of the given string.

In the following lines, we give a brief description of the partitioning algorithm used for solving the first problem mentioned.

For a factor w in x, the set of end positions of all the occurrences of w in x, gives us the *end set* of w. We define an equivalence relation $\approx_\ell$ on positions of x such that $i \approx_\ell j$ iff $x[i-\ell+1\,..\,i] = x[j-\ell+1\,..\,j]$. Therefore, depending on the length of the factor, we get equivalence classes for each length ℓ, $1 \leq \ell \leq n$. Equivalence classes for $\ell = 1$ are found by going over x once, and keeping the occurrences of each letter in separate sets. For larger ℓ, we consider classes of the previous level to make a refinement on them, and calculate the classes of that level. So on level ℓ, $1 < \ell \leq n$, we refine a class C by a class D, by splitting C in classes $\{i \in C : i-1 \in D\}, \{i \in C : i-1 \notin D\}$. In order to achieve a good running time, we do not use all classes for refinement; only classes of the previous level, which were split two levels before, are used. From

those, we can omit the largest siblings of each family, and use only the small ones for the computation. We terminate the algorithm when all classes reach a singleton stage.

In Figure 4.1, the classes of $\approx_\ell$, $\ell = 1, 2, \ldots, 8$, for the string $x =$ `abaababaabaabab` are illustrated. The partitioning algorithm creates two types of sets at each round of equivalent classes: the old and the new ones (see Fig. 4.1). A set is called *old*, if it has been created by deletion of elements from its parent set, i.e. it consists of remaining elements only. A set is called *new*, if it is composed from deleted elements from its parent set. Hence, the partitioning algorithm on x will give us distinct end sets E_i with their corresponding factor length range $(\ell_{\min_i}, \ell_{\max_i})$, as shown in the example below.

Example Let us consider $x =$ `abaababaabaabab`, the example string from Figure 4.1. Then, in level order, E_3 is the set $\{4, 9, 12\}$ with corresponding factor length range $(2, 4)$.

Proposition 4.1.10 ([154]) *The number of distinct end sets in the partitioning for a string of length n is $O(n)$.*

Proposition 4.1.11 ([75]) *The complexity of the partitioning algorithm for a string of length n is $O($number of elements of new sets$)$, where the number of elements of new sets in the partitioning is bounded above by $n \log n$.*

While executing the partitioning algorithm, we also maintain the following data structures:

- For each end set $E_i = \{a_1, a_2, .., a_k\}$, with corresponding factor length ℓ, of x, such that $k > 1$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x[i]$	a	b	a	a	b	a	b	a	a	b	a	a	b	a	b

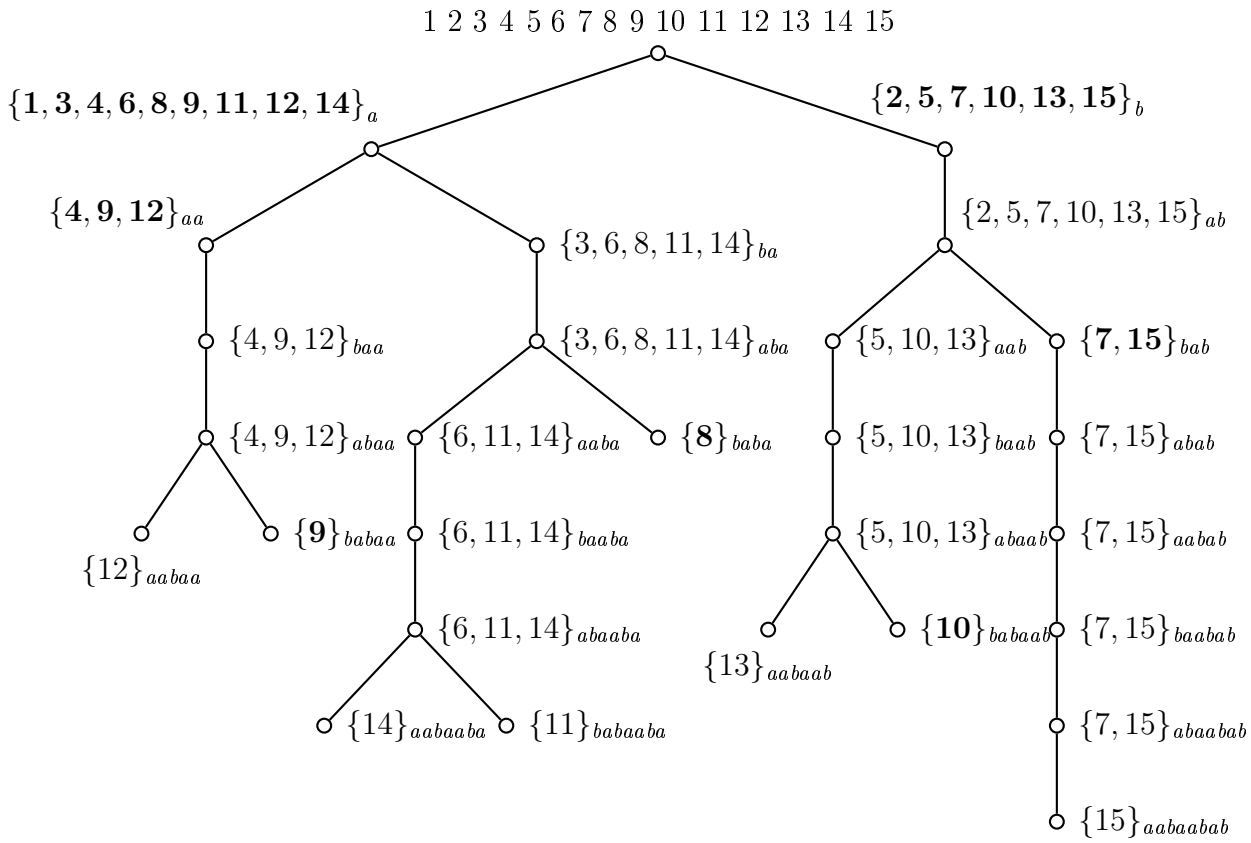


Figure 4.1: Classes of equivalence and their refinements for string $x =$ `abaababaabaabab`. The sets considered for the computation of the partitioning are shown in bold.

1. $Gap_{i,\ell}$ list

$$Gap_{i,\ell}(a_j) = a_{j+1} - a_j, j \in \{1, 2, \ldots, k-1\} \tag{4.5}$$

2. $cover_{i,\ell}$ value

   An end set E_i, with corresponding factor length ℓ, is said to have $cover_{i,\ell}$ value equal to the first element of the set, say a_j, such that $Gap_{i,\ell}(a_j) > \ell$, otherwise a value equal to the last element of the set. This value gives the last element coverable in each set by its equivalent factor, starting from the first

element of the set.

- For each distinct end set $E_i = \{a_1, a_2, .., a_k\}$, with corresponding factor length range $(\ell_{\min_i}, \ell_{\max_i})$, of x, such that $k > 1$.

  1. CV_i array

     An array CV_i, such that $\mathsf{CV}_i[j]$ is the j^{th} $cover_{i,\ell}$ value, for some $\ell_{\min_i} \leq \ell \leq \ell_{\max_i}$, appeared in E_i in ascending order.

  2. FL_i array

     An array FL_i, such that $\mathsf{FL}_i[j] = \ell$, where ℓ is the factor length of E_i when the j^{th} $cover_{i,\ell}$ value, for some $\ell_{\min_i} \leq \ell \leq \ell_{\max_i}$, first appears.

  3. $first_i$ value

     A value $first_i$, which gives the first element that has not been assigned a minimal right seed in the set or any of its ancestors.

Below, we show how to update the above mentioned data structures in time $O(n \log n)$.

- **In new sets:**

  When a new set of size m is created, we can update all relevant data structures in $O(m)$ time. By running over the elements of the set in order, we can easily update the *cover* value, *Gap* list, and the CV and FL arrays.

- **In old sets:**

  An old set is created from its parent set, by the deletion of d elements from it. When an element a_j is deleted from its parent set, we can easily update the *Gap* values of its neighbours. All these operations take time $O(d)$, where d is the number of elements of new sibling sets of the old set. Updating the *cover* value is more

difficult, as new data structures have to be created, but is still under control, as of Lemma 4.1.12, below. The CV and FL arrays are updated when a new *cover* value is found.

Lemma 4.1.12 *In a chain of old sets with a new set of size m as root, the cover values are calculated in the whole chain in $O(m)$ time.*

Proof We create a queue Q that initially includes the *Gap* list of the first old set in the chain. When a new gap is created by deletion of an element in subsequent old sets, two gaps are merged. Then, we stack the new gap on top of the second gap, that is used to form it in Q, and so on (those gaps have the same element on their right edge). We also keep pointers on the new gaps, formed in the corresponding *First-appear*(ℓ) list (those lists are kept in a different queue), where ℓ is the length of the corresponding factor of the set. Pointers for deleted gaps are added in the corresponding *Delete*(ℓ) list (those lists are kept in a different queue, as well).

We then go over the *First-appear* and *Delete* queues, which mark the beginning of a distinct old set. Gaps in the corresponding *First-appear*(ℓ) list are moved to the right of the element at their bottom, and gaps that are in the corresponding *Delete*(ℓ) list are deleted, maintaining the structure of Q. We then search for *cover* value in Q, by popping out gaps from Q, until the first gap in Q is greater than ℓ (*cover* value is the element on the left edge of the last gap considered), or Q becomes empty (*cover* value is the element on the right edge of the last gap considered). Whenever a gap, that has stacked elements on it, is popped out of Q, its stacked elements are passed to an element out of the queue called *start*, whose right element is taken to be the first gap in Q . If *cover* value found is smaller than next length to be encountered in *First-appear* queue, we check for more *cover* value changes as before.

It is easy to observe that Q has at most $2s-1$ elements, where s is the size of the *Gap* list of the first old set in the chain. Thus, we can create the lists *First-appear*, *Delete* and queue Q in $\mathcal{O}(m)$ time, as $s < m$. We check for *cover* value when a distinct old set is encountered for the first time (at most m such sets, i.e. sets of size $m, m-1, \ldots, 1$ made by removal of an element at each step). While inside that set, *cover* value changes are made iff the gap after a *cover* value gets equal to the factor length (overall at most $|Q|$ such cases, one for each element of Q). Failed attempts are made once when a distinct old set is encountered for the first time and at most once for each element of Q, therefore at most $|Q| + m$ failed attempts. □

Therefore, the maintenance of the above data structures takes time $O(\textit{number of elements of new sets})$, where the number of elements of new sets in the partitioning is bounded above by $n \log n$, as in Proposition 4.1.11.

Example Consider the example string from Fig. 4.1. Starting with the set for factor a, and continuing just with old sets (actually the first set in not exactly an old set here), we get the data structures in Fig. 4.2.

$\ell = 1$	1	3	4	6	8	9	11	12	14
$\ell = 2$		3		6	8		11		14
$\ell = 4$				6			11		14
$\ell = 7$									14

(a) End sets

$\ell = 1$	2	1	2	2	1	2	1	2
$\ell = 2$			3	2		3		3
$\ell = 4$						5		3

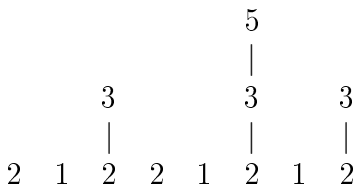
(b) *Gap* lists

Figure 4.2: End sets and corresponding *Gap* lists

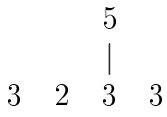
In Fig. 4.3(a), we show how queue Q will look for $\ell = 1$.

First-appear(1) keeps pointers at the bottom of the queue.

First-appear(2) keeps pointers at the elements stacked a level above them. *First-appear*(4) keeps pointers at the elements stacked two levels above them. In Fig. 4.3(b) and 4.3(c), we show how the queue Q will look for $\ell = 2$ and for $\ell = 4$, respectively.



(a) Queue Q for $\ell = 1$



(b) Queue Q for $\ell = 2$

5 3

(c) Queue Q for $\ell = 4$

Figure 4.3: Queue Q

We are now in a position to calculate *cover* values.

Before proceeding with calculating the minimal right-seed array of x, we also prove the following auxiliary lemma.

Lemma 4.1.13 *Let an end set $E_i = \{a_1, a_2, \ldots, a_k\}$ of x with corresponding factor z, $|z| = \ell$. Then z is a right seed for some set $\{x[1\,..\,a_1]$, $x[1\,..\,a_2], \ldots, x[1\,..\,a_s]\}$, where $1 \leq s \leq k$ and $s = 0$ gives us the empty set. There is no other prefix of y with its end position in E_i having z as a right seed.*

Proof Let a_t be the first element of E_i such that z is not a right seed of $x[0\,..\,a_t]$. If there exists no such element, then $s = k$ and z is a right seed for $\{x[0\,..\,a_1], x[0\,..\,a_2], \ldots x[0\,..\,a_k]\}$. If there exists no element $a_q \in E_i$ greater than a_t, such that z is a right seed of $x[0\,..\,a_q]$, then $s = t - 1$,

and the required set S is:

$$S = \begin{cases} \emptyset, & \text{if } s = 0 \\ \{x[1\,..\,a_1], x[1\,..\,a_2], .. x[1\,..\,a_{t-1}]\}, & \text{otherwise} \end{cases} \tag{4.6}$$

Suppose that there exists an element $a_q \in E_i$ greater than a_t such that z is a right seed of $x[1\,..\,a_q]$, i.e. it covers a superstring u, where $u = vx[1\,..\,a_q]$ and v is a possibly empty string. Therefore, there exists an occurrence of z in u ending in some $p \in \{x[a_t - \ell], \ldots, x[a_t - 1]\}$. Thus z is a cover of $vx[1\,..\,p]$. But there also exists an occurrence of z in x ending in $x[a_t]$. This shows that z is a cover of $vx[1\,..\,a_t]$, and hence a right seed of $x[1\,..\,a_t]$, which gives a contradiction. □

We are now in a position to calculate the minimal right-seed array of x by operating on the distinct end sets, while running over them in a level order traversal of the partition tree. The value *first* is passed from a parent set to the child set. If *first* value is in the set, we do not need to update the value. If *first* value is not in a new child set, it can be easily updated by searching for the smallest element which is greater or equal to *first* value in the new set; running over the elements of the set (Lemma 4.1.13), takes time $O(\textit{size of the new set})$. If *first* value is not in the old child set, we need to find the value just after it (Lemma 4.1.13), by searching in the elements of the parent set after *first* value; this takes time $\mathcal{O}(k)$, where k is the number of elements of new sibling sets of the old set. Keeping *first* value increases the time requirements for the partitioning algorithm only by a constant factor.

The period of $x[1\,..\,i]$, $\mathsf{P}[i]$, gives also the minimal right seed that can occur only once (that is why, in the next lines, we exclude distinct sets of size one; if they have not been assigned a right seed yet, then $\mathsf{P}[i]$ gives the length of their minimal right seed).

Let δ denote the first element of each distinct end set E_i, with corresponding factor length range $(\ell_{\min_i}, \ell_{\max_i})$, then the following hold.

- If $\mathit{first}_i \leq \mathit{cover}_{i,\ell_{\min_i}}$, by Lemma 4.1.2, the length of the minimal right seed for $y[1\,..\,\mathit{first}_i]$ is

$$max\{\ell_{\min_i}, \mathsf{P}[\mathit{first}_i] - (\mathit{first}_i - \delta)\} \tag{4.7}$$

  If there are no such lengths in the factor length range of E_i, we stop operations in that set (as a consequence of Lemma 4.1.13).

- If $\mathit{first}_i > \mathit{cover}_{i,\ell_{\min_i}}$, we move to the smallest factor length, denoted by γ, such that $\mathit{first}_i \leq \mathit{cover}_{i,\gamma}$. This is easily found using the corresponding arrays CV_i and FL_i. By Lemma 4.1.2, the length of the minimal right seed for $y[1\,..\,\mathit{first}_i]$ is

$$max\{\gamma, \mathsf{P}[\mathit{first}_i] - (\mathit{first}_i - \delta)\} \tag{4.8}$$

  If there are no such lengths in the factor length range of E_i, we stop operations in that set (as a consequence of Lemma 4.1.13).

If the minimal right seed of $x[1\,..\,\mathit{first}_i]$ is found, we assign the smallest element of E_i which is greater than first_i, as a new value for first_i, and continue searching from E_i with corresponding factor length the last length assigned as a minimal right seed length (as a consequence of Lemma 4.1.13).

Reporting minimal right seeds takes time $\Theta(n)$, i.e. one report for each position, as constant time is needed for each report. Failed attempts are made:

- at most once per report (when after the report, the next element of the set does not give a minimal right seed)

- at most twice per distinct set (one at the start of searching in the set, and one on a failure finding a suitable class for a future minimal right seed)

Therefore failed attempts are of $\mathcal{O}(n)$, and as constant time is needed for each report, the overall time needed for failed attempts is $\mathcal{O}(n)$. Also going over the *cover* changes takes time proportional to reporting *cover* values, which is $\mathcal{O}(n \log n)$.

Theorem 4.1.14 *Computing the minimal right-seed array of a given string x of length n can be done in $O(n \log n)$ time.*

Proof By Proposition 4.1.11, executing the partitioning algorithm takes time $O(n \log n)$. Maintaining the *Gap* list and the value *cover* increases the time requirements for the partitioning algorithm by a constant factor. Maintaining the array CV and the array FL is of no extra cost to maintaining the value *cover*. Maintaining the value *first* also increases the time requirements for the partitioning algorithm by a constant factor. The KMP algorithm, used for computing the period array P, runs in linear time [165]. Reporting the minimal right seeds requires $\Theta(n)$ time, i.e. one report for each position. Failed attempts are at most $O(n)$. Going over the *cover* changes, while searching for minimal right seeds, takes time proportional to reporting *cover* values, which is $O(n \log n)$. Hence, overall, the described algorithm runs in $O(n \log n)$ time. □

Surprisingly the algorithm for computing the minimum right seed array is of higher time complexity than the algorithm for computing the minimum left seed array. This is due to the fact that all left seeds have common prefixes, they are built on a previous one, while right seeds do not have a common start and so they grow independently.

Example An overview of the algorithm for computing the minimal right -seed array of string $x = \texttt{abaababaabaabab}$ is shown in Fig. 4.4.

It is easy to see that the problem of finding the maximal right-seed array is solved similarly to the problem of computing the maximal right-seed array (observe the similarity of Lemma 4.1.6 and Lemma 4.1.7), and so we obtain the following result.

Theorem 4.1.15 *Computing the maximal right-seed array of a given string x of length n can be done in linear time.*

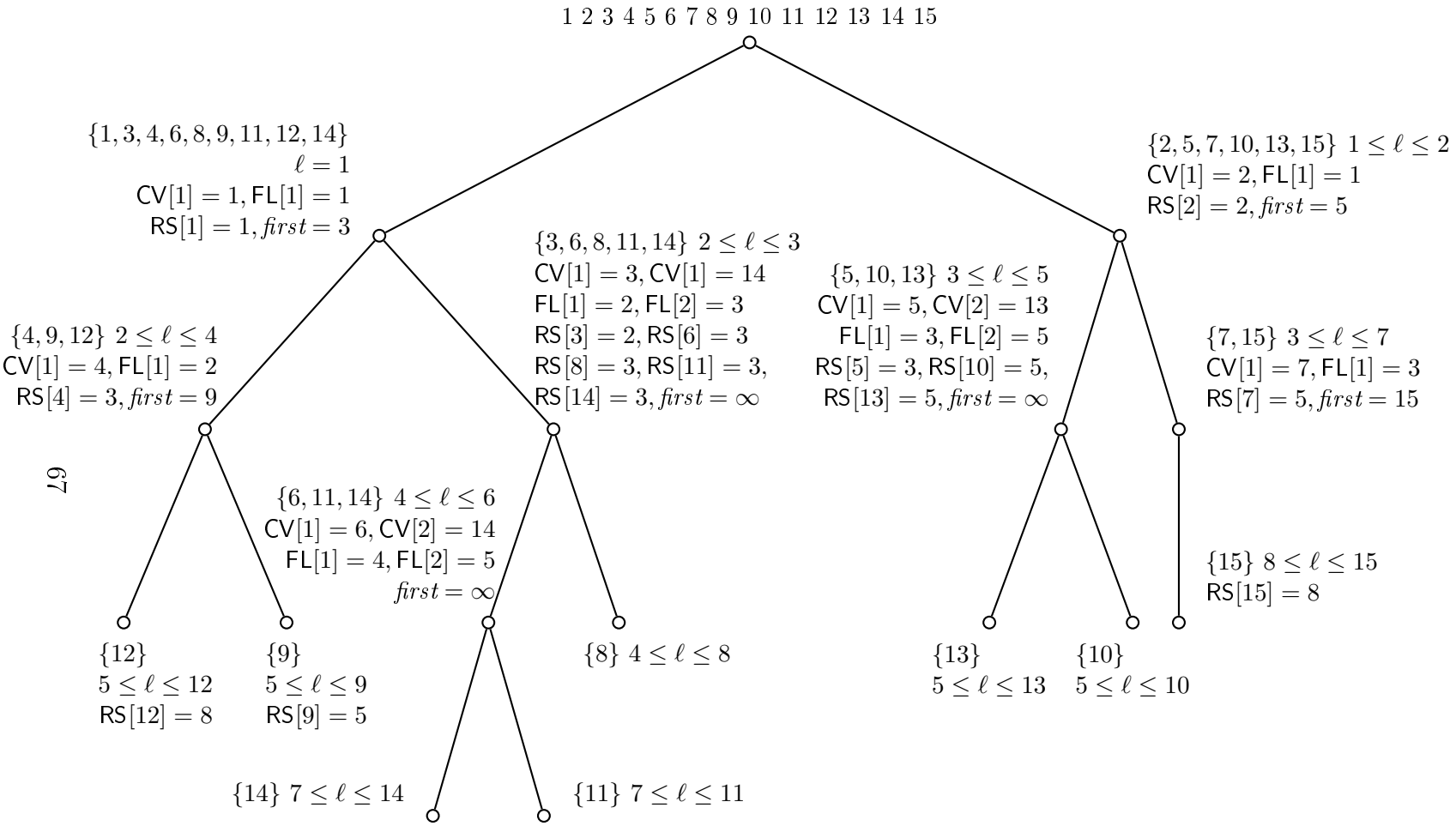


Figure 4.4: Computing the minimal right-seed array of string $x =$ `abaababaabaabab` (see also Fig. 4.1)

4.2 On the appearance of seeds in words

4.2.1 Average case

In this section we study the behaviour of the average number of seeds in a word of Σ^n, the set of words of length n. This number is given by the expected value of the number of seeds when we consider a word $x = x[1\,..\,n]$ with all letters of x drawn independently from $\Sigma = (a_1, a_2, \ldots, a_\sigma)$ with equal probability $\frac{1}{\sigma}$ for each letter.

We show that a word of length n has $O(n)$ seeds on average by using some combinatorial properties and series relations. However, before that, we need to prove the following basic lemma:

Lemma 4.2.1 *A seed of length ℓ which appears only once in x implies the appearance of either a periodic prefix or a periodic suffix of x of length at least $\frac{n+\ell}{2}$ and period at most ℓ.*

Proof Let $s = x[i\,..\,i+\ell-1]$ be a non-trivial seed of x occurring only once in x. Therefore, $x = usv$ with $0 \leq |u| < \ell$ and $0 \leq |v| < \ell$, where not both u and v are empty strings.

Let's assume first that both $|v| > 0$ and $|u| > 0$. Then a prefix of s must occur after position i, say at $i + j$ $(1 \leq j \leq \ell)$, and consequently x can be written as $x = u(s[1\,..\,j])^{\frac{n-|u|}{j}}$. Hence the suffix of x of length $n-|u|$ is periodic with period at most ℓ. Similarly, a suffix of s must end before position $i+\ell-1$, yielding a periodic prefix of x of length $n-|v|$ and period at most ℓ. As $|u|+|v| = n-\ell$ either $n-|u|$ or $n-|v|$ is at least $\frac{n+\ell}{2}$.

If only $|u| = 0$ or only $|v| = 0$ then x has period at most ℓ and our requirements are met. □

In a similar manner, we can prove the following lemma.

Lemma 4.2.2 *A seed $y = x[i..i+\ell-1]$ of length ℓ, where $1 \leq i \leq n-\ell+1$ and $1 \leq \ell \leq n-1$, which appears at least twice in x implies the appearance of a square of form $x[i..i+2\ell-1] = yy$ or $x[i-\ell..i+\ell-1] = yy$ or the appearance of a quasiperiodic square of form $x[i..i+\ell+k-1] = yv$ or $x[i-k..i+\ell-1] = vy$, where $1 \leq k \leq \ell-1$ and v is a substring of x such that $|v| = k$, in x.*

Proof As there exist at least two occurrences of y in x and y is a cover of a superstring of x then there should be a y starting at position $j \in \{i+1, \ldots, i+\ell\}$ or ending at position $j \in \{i-1, \ldots, i+\ell-2\}$. Those two occurrences form the required square (if they are consecutive) or quasiperiodic square (if they overlap). □

The following lemma regarding the polylogarithm function will also be required to prove further results.

Lemma 4.2.3 *[236] For $z \in \mathbb{C}$ such that $|z| < 1$, $Li_{-1}(z) = \sum_{k=1}^{+\infty} kz^k = \frac{z}{(1-z)^2}$.*

Using the above lemmas, we are now able to prove the main result of this section.

Theorem 4.2.4 *On average a word of length n has $O(n)$ seeds.*

Proof Let x be a word of length n, with its letters drawn independently from $(a_1, a_2, \ldots, a_\sigma)$ with a constant probability distribution $(\frac{1}{\sigma}, \ldots, \frac{1}{\sigma})$.

$$E(\text{number of seeds in } x) = E(\#\text{ of seeds appearing only once in } x) + E(\#\text{ of seeds appearing at least twice in } x)$$

As of Lemma 4.2.1 we get:

$$
\begin{aligned}
&E(\text{number of seeds that appear only once in } x)\\
&= \sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1} P(x[i\,..\,i+\ell-1] \text{ is a seed that appears only once in } x)\\
&\le \sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1} P(x \text{ has a periodic prefix or a periodic suffix of length}\\
&\text{at least } \frac{n+\ell}{2} \text{ and period at most } \ell)\\
&\le 2\sum_{\ell=1}^{n}\sum_{i=1}^{n} P(x \text{ has a periodic prefix of length at least } \frac{n+\ell}{2} \text{ and}\\
&\text{period at most } \ell)\\
&\le 2n\sum_{\ell=1}^{n}\sum_{k=1}^{\ell}\frac{\sigma^k}{\sigma^{\frac{n+\ell}{2}}} \le \frac{2n}{\sigma^{\frac{n}{2}}}\sum_{\ell=1}^{n}\frac{1}{\sigma^{\frac{\ell}{2}}}\frac{\sigma(\sigma^\ell-1)}{\sigma-1}\\
&\le \frac{2\sigma n}{(\sigma-1)\sigma^{\frac{n}{2}}}\sum_{\ell=1}^{n}(\sigma^{\frac{\ell}{2}}-\sigma^{-\frac{\ell}{2}})\\
&\le \frac{2\sigma n}{(\sigma-1)\sigma^{\frac{n}{2}}}\left(\frac{\sigma^{\frac{1}{2}}(\sigma^{\frac{n}{2}}-1)}{\sigma^{\frac{1}{2}}-1}-\frac{\sigma^{-\frac{1}{2}}(\sigma^{-\frac{n}{2}}-1)}{\sigma^{-\frac{1}{2}}-1}\right)\\
&= \frac{2\sigma^{\frac{3}{2}}}{(\sigma-1)(\sqrt{\sigma}-1)}n+o(n)
\end{aligned}
$$

As of Lemma 4.2.2 we get:

$$
\begin{aligned}
&E(\text{number of seeds that appear at least twice in } x)\\
&= \sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1} P(x[i\,..\,j] \text{ is a seed that appears at least twice in } x)\\
&\le \sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1}(\sum_{k=1}^{\ell} P(x[i\,..\,i+\ell-1+k] \text{ is a quasiperiodic square})\\
&\quad + P(x[i\,..\,i+2\ell-1] \text{ is a square}) + \sum_{k=1}^{\ell} P(x[i-k\,..\,i+\ell-1] \text{ is a}\\
&\quad \text{quasiperiodic square}) + P(x[i-\ell\,..\,i+\ell-1] \text{ is a square}))\\
&\le 2\sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1}\sum_{k=1}^{\ell-1}\frac{\sigma^k}{\sigma^{\ell+k}} + 2\sum_{\ell=1}^{n-\ell+1}\sum_{i=1}^{n}\frac{\sigma^\ell}{\sigma^{2\ell}} = 2\sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1}\sum_{k=1}^{\ell}\frac{\sigma^k}{\sigma^{\ell+k}}
\end{aligned}
$$

$$= 2\sum_{\ell=1}^{n}\sum_{i=1}^{n-\ell+1}\sum_{k=1}^{\ell}\frac{1}{\sigma^{\ell}} = 2\sum_{\ell=1}^{n}\frac{n-\ell+1}{\sigma^{\ell}} = 2(n+1)\sum_{\ell=1}^{n}\frac{1}{\sigma^{\ell}} - 2\sum_{\ell=1}^{n}\frac{\ell}{\sigma^{\ell}}$$
$$= 2(n+1)\frac{(1-\sigma^{-n})}{\sigma-1} - 2\sum_{\ell=1}^{n}\frac{\ell}{\sigma^{\ell}}$$

Lemma 4.2.3 suggests that the last series are bounded by constant terms, thus proving the theorem. □

4.2.2 Maximum number of distinct seeds in a word

It is easy to see that there exist words in which every factor is a seed, e.g. a^n. In this section we are investigating how many distinct seeds can appear in a word. We denote the maximum number of distinct seeds in a word of length n by $Seeds(n)$.

Lower Bound

Fibonacci words provide us with a first lower bound.

Lemma 4.2.5 *There exists an infinite family of words for which* $Seeds(n) \geq \frac{\phi^2+1}{2\phi^6}n^2 + o(n^2)$*, where* ϕ *is the golden ratio.*

Proof In [62, 63] it was shown that for Fibonacci words it holds that:

$$\lim_{n\to+\infty}\frac{Seeds(F_n)}{|F_n|^2} = \frac{\phi^2+1}{2\phi^6} = 0.100813061875578\ldots$$

□

Next we show that periodic words are quite rich in word regularities. In particular, words that are squares have more seeds than Fibonacci words.

Lemma 4.2.6 $Seeds(n) \geq \frac{n^2}{8} + o(n^2)$.

Proof When n is even we consider the word $(a_1a_2\dots a_{\frac{n}{2}})^2$. Obviously every factor of length greater than $\frac{n}{2}$ is a seed of the word. There are $\frac{n}{2}$ such factors starting from the first position in the word, $\frac{n}{2}-1$ such factors starting from the second position of the word and so on. Overall:

$$\sum_{i=1}^{\frac{n}{2}} i = \frac{\frac{n}{2}(\frac{n}{2}+1)}{2} = \frac{n^2}{8}+\frac{n}{4}$$

Similarly, when n is odd we consider the word $(a_1a_2\dots a_{\frac{n-1}{2}})^2 a_{\frac{n+1}{2}}$ which yields $\frac{n^2}{8}+\frac{n-1}{2}$ distinct seeds. □

In the following lemma, we prove that among all words having similar structure to the squares considered in the proof of the previous lemma (i.e. words of form $(a_1a_2\dots a_{|p|})^c = p^c$, where $c|p| = n$ and $1 \leq |p| < n$), cubes or words close to being cubes (i.e. words in which $|p| = \frac{n}{3} + o(n)$) achieve maximum number of distinct seeds.

Lemma 4.2.7 *$Seeds(n) \geq \frac{1}{6}n^2 + o(n^2)$.*

Proof We consider the word $(a_1a_2\dots a_{|p|})^c = p^c$, where $c|p| = n$ and $1 \leq |p| < n$. Obviously every factor of length greater than $|p|$ or equal to $|p|$ is a seed of the word. There are $n-|p|+1$ such factors starting from the first position in the word, $n-|p|$ such factors starting from the second position of the word, etc. Due to factors repeating, we only consider factors starting from the first period of the word. We distinguish two cases, according to $|p|$.

For $\frac{n}{2} \leq |p| < n$ we get the following number of distinct seeds:

$$\begin{aligned}\sum_{i=1}^{n-|p|+1} i &= \frac{(n-|p|+1)(n-|p|+2)}{2} = \frac{(n-\frac{n}{c}+1)(n-\frac{n}{c}+2)}{2} \\ &= \frac{(1-\frac{1}{c})^2}{2}n^2 + o(n^2)\end{aligned}$$

Having assumed that $\frac{n}{2} \leq |p| < n$, it follows that $1 < c \leq 2$ and the above expression maximizes for $c = 2$, giving $\frac{1}{8}n^2 + o(n^2)$ different seeds.

For $1 \leq |p| < \frac{n}{2}$ we get the following number of distinct seeds:

$$\begin{aligned}\sum_{i=n-2|p|+2}^{n-|p|+1} i &= \frac{(n-|p|+1)(n-|p|+2)}{2} - \frac{(n-2|p|+1)(n-2|p|+2)}{2} \\ &= \frac{(n-\frac{n}{c}+1)(n-\frac{n}{c}+2)}{2} - \frac{(n-2\frac{n}{c}+1)(n-2\frac{n}{c}+2)}{2} \\ &= \frac{(1-\frac{1}{c})^2-(1-\frac{2}{c})^2}{2}n^2 + o(n^2) = \frac{\frac{2}{c}-\frac{3}{c^2}}{2}n^2 + o(n^2)\end{aligned}$$

As $2 < c \leq n$ the above expression maximizes for $c = 3$, giving $\frac{1}{6}n^2+o(n^2)$ different seeds. Even when $c = 3$ cannot be achieved, choosing $p = \lfloor \frac{n}{3} \rfloor$ gives the required bound. □

Upper Bound

A first upper bound is given using the restriction that a seed must be a factor of the word.

Lemma 4.2.8 *The number of distinct seeds of a non-empty word x is at most $\frac{n(n+1)}{2}$, where $|x| = n$.*

Proof The number of non-empty factors of x is $n + \binom{n}{2} = \frac{n(n+1)}{2}$. □

In the following lemma we prove a better upper bound using more combinatorial properties.

Lemma 4.2.9 *$Seeds(n) \leq \frac{1}{4}n^2 + o(n^2)$.*

Proof We first consider n to be even. A seed of length ℓ, $n/2+1 \leq \ell \leq n$, appears at least once in x (as it is a factor of x). Therefore, x has at most $n - \ell + 1$ seeds of length ℓ. Overall:

$$\sum_{\ell=n/2+1}^{n} (n-\ell+1) < \sum_{i=1}^{n/2} i = \frac{1}{8}n^2 + o(n^2)$$

A seed of length ℓ, $1 \leq \ell \leq n/2$, has at least one starting position in $x[1\,..\,\ell]$ (as it covers x). Therefore, x has at most ℓ different seeds of length ℓ. Overall:

$$\sum_{\ell=1}^{n/2} \ell = \frac{1}{8}n^2 + o(n^2)$$

By summing the two expressions we get $\frac{1}{4}n^2 + o(n^2)$ as an upper bound. The proof for odd n is similar. □

Structure of the extremal word

The bounds revealed in the previous subsections help us to get some information for a word that maximizes $Seeds(n)$. In particular, we are able to restrict the length of repeating factors and the period and exponent of a run in such a word.

Theorem 4.2.10 *A word x of length n having the maximum number of seeds $Seeds(n)$ contains no repeating factors of length $cn + o(n)$, where $\sqrt{\frac{2}{3}} < c \leq 1$.*

Proof For large n, a repeating factor of length cn would mean more than $\frac{1}{3}n^2$ repeats (the subwords of the factor) and hence less than $\frac{1}{2}n^2 - \frac{1}{3}n^2 = \frac{1}{6}n^2$ candidate seeds, contradicting Lemma 4.2.7. □

Theorem 4.2.11 *There are no runs of period p and exponent c in a word achieving the upper bound for which $\frac{1}{2}c^2p^2 - \frac{1}{2}cp + \frac{3}{2}p^2 > \frac{1}{3}n^2$.*

Proof A run of period p and exponent c in x gives at least

$$\frac{1}{2}(cp)(cp+1) - \sum_{i=cp-2p+2}^{cp-p+1} i$$

repeating factors. The dominating terms of this expression are $\frac{1}{2}c^2p^2 -$

$\frac{1}{2}cp + \frac{3}{2}p^2$. Therefore, as in Theorem 4.2.7, for large n we have:

$$\frac{1}{2}n^2 - \left(\frac{1}{2}c^2p^2 - \frac{1}{2}cp + \frac{3}{2}p^2\right) \geq \frac{1}{6}n^2$$

□

We conclude by showing the existing best bounds for the maximum number of distinct regularities in a word in the following table:

Regularity	**Maximum number in a word**
Runs	Between $1.048n$ [82] and $0.944575712n$ [220]
Cubic runs	Between $0.5n$ and $0.406n$ [83]
Squares	Between $n - o(n)$ [117] and $2n - \Theta(logn)$ [150]
Cubes	Between $\frac{n}{2}$ and $\frac{4n}{5}$ [172]
Seeds	Between $\frac{n^2}{6} + o(n^2)$ and $\frac{n^2}{4} + o(n^2)$

4.3 Overlapping factors in words

4.3.1 An algorithm for the computation of all overlapping factors of a word

In this section we show the relation between runs and overlapping factors of a word, thus giving a linear time algorithm for their computation.

The following lemma sheds some light on the periodic structure of quasiperiodic squares.

Lemma 4.3.1 *A factor w of a string x is a quasiperiodic square if and only if $Period(w) < \frac{|w|}{2}$.*

Proof ($\Rightarrow$) Let $w = x[i\,..\,j] = z^k$, where $k > 2$. Then $w = z^{k-1}v = uz^{k-1}$, where z^{k-1} and v are non empty words and $|z^{k-1}| > |v| = |z|$ and hence w is a quasiperiodic square.

($\Leftarrow$) Let $w = x[i\,..\,j] = yu = vy$, where y, u and v are non empty words and $|y| > |v|$. As both y and v are prefixes of w and $|y| > |v|$ then v is also a prefix of y. Therefore $w = yu = vy = vvp$, where p is a non empty word. Using similar arguments $w = v^{\frac{|w|}{|v|}}$, where $\frac{|w|}{|v|} > 2$. □

Runs are maximal periodicities, therefore by knowing all the runs of a word we can identify all its periodic factors, its quasiperiodic squares and overlapping factors. Therefore, the following lemma will prove quite useful in the identification of all overlapping factors in a word.

Lemma 4.3.2 *[168] There exists a linear time algorithm that identifies all runs in a string x.*

Using the above lemma we are now in a position to describe a linear time algorithm for the identification of all overlapping factors in a word.

Theorem 4.3.3 *There exists a linear time algorithm that identifies all overlapping factors in a string x.*

Proof Immediate consequence of Lemma 4.3.1 and Lemma 4.3.2.

Any quasiperiodic square of period $|z|$ that appears in y implies the appearance of a run of period $|z|$ in y which must include it and vice versa. Hence, suppose $y[i\,..\,j] = z^k$ is a run, then every factor of $y[i\,..\,j]$ of length ℓ, where $2|z| < \ell \leq j - i + 1$, is a quasiperiodic square with period at most $|z|$. If $y[k\,..\,k+\ell-1]$ is such a quasiperiodic square then $y[k\,..\,k+\ell-|z|-1]$ and $y[k+|z|\,..\,k+\ell-1]$ are some of its overlapping factors. Any longer ones imply the appearance of a run with a shorter period (and will be identified with its help). Suppose $y[k\,..\,k+m-1]$ and $y[k+\ell-m\,..\,k+\ell-1]$, with $0 < m < \ell - |z|$, are shorter overlapping factors of this quasiperiodic square. Then $y[k\,..\,k+m-1]$ and $y[k+\ell-m\,..\,k+\ell-1]$ reappear as a prefix of $y[k+|z|\,..\,k+\ell-1]$ and a suffix of $y[k\,..\,k+\ell-|z|-1]$ respectively. Hence, they are overlapping factors of the

quasiperiodic squares $y[k\,..\,k+m+|z|-1]$ and $y[k+\ell-m-|z|\,..\,k+\ell-1]$ respectively which are indicated by this run. □

4.3.2 Bounds on the maximum number of distinct overlapping factors in a word

In this section we investigate the maximum number of overlapping factors in a word. We show a simple upper bound by considering basic combinatorial properties. Fibonacci words give a first lower bound. Then we provide a more complex example, giving a better lower bound.

Theorem 4.3.4 *The number of distinct overlapping factors in a word y, denoted by $OF(y)$ is bounded above by $\frac{n^2-n}{4}$, where $|y| = n$.*

Proof The number of factors of y of length at least 2 is $\binom{n}{2}$. When a factor overlaps with another it means that it appears at least twice in y. Therefore, $OF(y) \leq \frac{1}{2}\binom{n}{2} = \frac{n^2-n}{4}$ □

Fibonacci words give a first lower bound. It is easy to observe that periodicity causes the appearance of many overlapping factors. The following example makes use of that observation to give a better lower bound on $OF(y)$. Using simple calculus one can verify that a period of $\frac{|y|}{5}$ gives maximum number of overlapping factors. We then allow periods to have similar structure to get a higher lower bound as shown in the following theorem.

Theorem 4.3.5 *The maximum number of distinct overlapping factors in a word y over the square of its length $|y| = n$, i.e. $\frac{OF(y)}{n^2}$, is at least $\frac{5}{48} = 0.1041666667\ldots$.*

Proof Consider the following family of words (see also Figure 4.5):

$$y_i = y_{i-1}^4 y_{i-1}[1\,..\,|y_{i-1}|-1]a_i \quad \text{with } y_0 = a_0$$

$$
\begin{aligned}
y_0 &= a_0 \\
y_1 &= \underbrace{a_0}_{y_0}\underbrace{a_0}_{y_0}\underbrace{a_0}_{y_0}\underbrace{a_0}_{y_0}a_1 \\
y_2 &= \underbrace{a_0a_0a_0a_0a_1}_{y_1}\underbrace{a_0a_0a_0a_0a_1}_{y_1}\underbrace{a_0a_0a_0a_0a_1}_{y_1}\underbrace{a_0a_0a_0a_0a_1}_{y_1}\underbrace{a_0a_0a_0a_0}_{y_1[1\,..\,|y_1|-1]}a_2 \\
&\vdots
\end{aligned}
$$

Figure 4.5: The family of words described in Theorem 4.3.5

Then:

$$\lim_{i\to+\infty}\frac{\mathrm{OF}(y_i)}{|y_i|^2} = \lim_{i\to+\infty}\frac{1}{n^2}(\sum_{j=1}^{i}\text{number of overlapping factors with length } \ell, \text{ where } |y_{j-1}| \le \ell < |y_j|)$$

In order to search for overlapping factors with length ℓ, where $|y_{j-1}| \le \ell < |y_j|$, it is enough to restrict our attention to the word y_j (as we can't include $y_j[|y_j|]$ in such a factor). Then counting such factors from the second occurrence of y_{j-1} in y_j we get $3|y_{j-1}|-1$ factors starting from position $y_j[|y_{j-1}|+1]$, $3|y_{j-1}|-2$ factors starting from position $y_j[|y_{j-1}|+2]$, ... and $2|y_{j-1}|$ factors starting from position $y_j[2|y_{j-1}|-1]$. Overall:

$$
\begin{aligned}
\lim_{i\to+\infty}\frac{\mathrm{OF}(y_i)}{|y_i|^2} &= \lim_{i\to+\infty}\frac{1}{n^2}\sum_{j=1}^{i}(\sum_{k=2|y_{j-1}|}^{3|y_{j-1}|-1}k) \\
&= \lim_{i\to+\infty}\frac{1}{n^2}\sum_{j=1}^{i}(\sum_{k=1}^{3|y_{j-1}|-1}k - \sum_{k=1}^{2|y_{j-1}|-1}k) \\
&= \lim_{i\to+\infty}\frac{1}{n^2}\sum_{j=1}^{i}\frac{5|y_{j-1}|^2-|y_{j-1}|}{2} \\
&= \lim_{i\to+\infty}\frac{1}{2n^2}(\sum_{j=1}^{i}5(\frac{n}{5^{i-j+1}})^2 - \sum_{j=1}^{i}\frac{n}{5^{i-j+1}})
\end{aligned}
$$

$$= \lim_{i\to+\infty} \frac{5}{2n^2} \sum_{j=1}^{i} (\frac{n}{5^{i-j+1}})^2 = \lim_{i\to+\infty} \frac{5}{2} \sum_{j=1}^{i} \frac{1}{25^{i-j+1}}$$

$$= \lim_{i\to+\infty} \frac{5}{2} \sum_{j=1}^{i} \frac{1}{25^j} = \frac{5}{2} \frac{1/25}{24/25} = \frac{5}{48} = 0.1041666667\ldots$$

□

Chapter 5

Abelian periodicity

Much research has been concentrated on classical periods, e.g. algorithms for finding all periods of a string, algorithms for the computation of the period array of a string [165], etc. Abelian periods are more flexible than classical ones and are defined in terms of Parikh vectors as in [74]. Relevant research has concentrated on their combinatorial properties, on algorithms for their efficient computation and on abelian borders (the complementary notion of the period). For a detailed survey of the area and also on some related applications see [48].

Abelian periodicity has been extensively studied over the last years [13, 14, 26, 28, 45, 87, 94, 217]. In 2006 Constantinescu and Ilie [74] proved a variant of Fine and Wilf's theorem for abelian periods, later extended for abelian periods in partial words [28]. Furthermore Avgustinovich et al. [14] attempted to give an abelian analogue of the Critical Factorization Theorem. Later, Fici et al. [113] investigated the appearance of abelian repetitions in sturmian words. Early efficient algorithms for abelian pattern matching were given in [100, 101] and later some linear time algorithms have been designed in [38, 39, 71]. Recently Fici et al. gave five algorithms for the computation of all abelian periods of a word [114]. They have proposed two offline algorithms, a brute force

algorithm and one that uses a select array, that run in $O(n^2\sigma)$ and three online algorithms, where the first two run in $O(n^3\sigma)$ and the other one runs in $O(n^3\sigma \log n)$. Experimentally the off line algorithm that makes use of the select array is said to be the fastest in practice. Finally, in a recent paper, Fici et al. [115] gave a quasi-linear time algorithm for the computation of a special type of abelian periods of a word together with some other algorithms.

Erdős [104] has proposed the problem of constructing an infinite word on an alphabet as small as possible such that the word does not contain any abelian squares. Pleasants [201] gave a construction on 5 letters until more recently Keränen [164] came with a construction on 4 letters. Furthermore Cummings and Smyth [86] showed a $\Theta(n^2)$ algorithm for the computation of all abelian squares of a word. Since then a lot of research has been concentrated around similar problems and their generalisations ([13, 14, 26, 45, 87, 94, 217]).

Richmond and Shallit [207] counted the number, $f_\sigma(n)$, of different abelian squares of length n over an alphabet of size σ, using basic combinatorial identities and gave an asymptotic estimate of this quantity:

$$f_\sigma(n) \sim \frac{\sigma^{2n+\frac{\sigma}{2}}(4\pi n)(1-\sigma)}{2}$$

Callan [41] presented a bijection between Barrucand deals and abelian squares, by showing a bijection between Barrucand deals and abelian matrices (a representation of an abelian square). We remind the reader how a Barrucand n-deal is formed: we start with a deck of $3n$ cards, n coloured red, n coloured green and n coloured blue, in denominations 1 through n,we then choose an arbitrary subset of the denominations and deal all cards of the chosen denominations into 3 equal-size hands to players designated red, green and blue in such a way that no player

receives a card of his own color. In the same paper the Barrucand deal is then related to other special numbers and combinatorial identities, thus revealing their connections to abelian squares. Domaratzki and Rampersad [94] showed that the set of abelian primitive strings is not context-free (i.e. that language can not be formed by only using rules or form $v \to w$, where v is a non terminal symbol and w is a set of terminal and non terminal symbols). Furthermore, contrary to the classical case, they proved that a string can have more than one abelian root and they gave some bounds on that number. Regarding the problem of determining whether a string is abelian primitive, they showed a worst case linear time algorithm involving prime factorisation. Karaman [161] showed a worst case linear time algorithm for the computation of all weak repetitions (abelian powers) in a Sturmian string by introducing a special encoding for the output. Later, Richomme et al. [208] showed that all Sturmian strings are everywhere abelian k-repetitive for all integers $k \geq 1$, i.e. every sufficiently long factor of the string has an abelian k^{th} power as a prefix. Furthermore, given a natural number k they proved that any position of a Sturmian string t starts with an abelian k^{th} power with abelian period ℓ_1 or ℓ_2. In the same context, Avgoustinovich et al. [14] showed that Sturmian strings have bounded right k-powers for every k, and they do not have bounded central powers. It is also interesting that the Fibonacci string, itself a Sturmian string, begins with arbitrarily high abelian powers [88]. Regarding the Thue-Morse word, Avgoustinovich et al. [14] proved that it has bounded abelian squares centered at every position in it and bounded abelian cubes to the right of every position in it. However there are no bounded abelian 4-powers centered or to the right of every position of the word. Finally, Fraenkel et al. [119, 120] studied the appearance of abelian squares in circular binary strings. They proved that the longest string with only k distinct

abelian circular squares contains $4k+2$ bits and has the form $(01)^{2k+1}$ or its complement. Cummings et al. [86] showed a rather straight-forward $\Theta(n^2)$ algorithm for the computation of all abelian squares of a string, the only one at the moment.

In this section, we show two $O(n^2)$ algorithms for the computation of all abelian periods of a string y [60], an improvement to the algorithms in [114], where the fastest one runs in $O(n^2\sigma)$ time. Furthermore our algorithms use $O(n)$ space in contrast to the $O(n^2)$ space algorithms in [114]. The first of our algorithms maps each letter to a suitable number such that each factor of the string can be identified by the unique sum of the numbers corresponding to its letters. The other one maps each letter to a prime number such that each factor of the string can be identified by the unique product of the numbers corresponding to its letters. We are then able to perform the required checks of Parikh vectors, necessary to identify abelian periods, with just one operation. Additionally we define weak abelian periods on strings and give an $O(n \log n)$ algorithm for their computation [60]. Some other algorithms for basic problems on identification of periods which form the basis of the previous ones are also analyzed [60]. We show a linear time algorithm for the computation of all abelian borders of a word x, which identifies the abelian borders of x by keeping track of the difference between Parikh vectors of prefixes and suffixes of the word of the same length [49]. We also comment on the appearance of abelian borders in Thue-Morse words [49]. Finally we prove that the average length of abelian borders of a word x is $\frac{n}{2}$, if it exists, and also that a binary word of length n has $\Theta(\sqrt{n})$ abelian borders on average in order to reveal more about their occurrence in words [49]. We also investigate the number of binary words whose shortest border has a given length, by identifying relations with Dyck words. Next, we give some bounds on the number of abelian border-free words of a given

length and on the number of abelian words of a given length that have at least one abelian border. Finally we provide an algorithm that finds the shortest abelian border of a non-abelian-border-free binary word in time $\Theta(\sqrt{n})$ on the average.

5.1 Identifying all abelian periods of a word and relevant problems

5.1.1 Definitions and Problems

In this section we formally define the problems that we are considering. We first give some introductory problems, which will help us to solve more complex problems that are defined later.

Before proceeding we need to introduce some more definitions.

Given a mapping $p : \Sigma \to A$, where A is the set of the first σ prime numbers, such that $p(\Sigma_i) = i_{th}$ prime number, the *P-signature* of a word y is defined to be equal to $\prod_{i=0}^{|y|-1} p(y[i])$. We remind the reader that a prime is a positive integer greater than 1 having exactly one positive divisor other than 1.

Given a mapping $s : \Sigma \to B$, where B is the set of the first $\sigma - 1$ powers of $n + 1$ and 0, such that:

$$s(\Sigma_i) = \begin{cases} 0, & i = \{1\} \\ (n+1)^{(i-2)}, & \text{otherwise,} \end{cases} \tag{5.1}$$

, where the *S-signature* of a word y is defined to be equal to $\sum_{i=0}^{|y|-1} s(y[i])$.

The array Pr, where $\mathsf{Pr}[i] = \prod_{j=0}^{i} p(y[j])$, is useful in computing the *P-signature* of substrings of y, as:

$$P\text{-}signature(y[q\,..\,k]) = \begin{cases} \mathsf{Pr}[k]/\mathsf{Pr}[q-1], & q \neq 0 \\ \mathsf{Pr}[k], & q = 0 \end{cases} \tag{5.2}$$

The array LS, where $\mathsf{LS}[i] = \sum_{j=0}^{i} s(y[j])$, is useful in computing the *S-signature* of substrings of y, as:

$$S\text{-}signature(y[q\,..\,k]) = \begin{cases} \mathsf{LS}[k] - \mathsf{LS}[q-1], & q \neq 0 \\ \mathsf{LS}[k], & q = 0 \end{cases} \tag{5.3}$$

We consider the following problems:

Problem 1 (Abelian period decision) *Decide if (h, p), where $0 \leq h < min(p, \lfloor \frac{n-1}{2} \rfloor + 1)$ and $1 \leq p \leq n$, is an abelian period of some string y.*

Problem 2 (String-Abelian period decision) *Decide if a string x, where $|x| = m < n$, composed from the same alphabet Σ as a string y can be an abelian period of y, i.e. there exist an abelian period (h, p) of y such that $y[h\,..\,h+p-1]$ is a permutation of x.*

Problem 3 (String-Abelian periods) *Output all abelian periods (h, p) of y such that $y[h\,..\,h+p-1]$ is a permutation of a string x, where $|x| = m < n$ and x is composed from the same alphabet Σ as y.*

Problem 4 (Computing all weak abelian periods of a string) *Compute all weak abelian periods of some string y.*

Problem 5 (Computing all abelian periods of a string) *Compute all abelian periods of some string y.*

5.1.2 Properties

In this section, we prove some useful properties for abelian periods and we also quote some fundamental properties of primes that prove to be useful for the analysis of our algorithms.

The following lemmas and theorems are related to prime numbers and they will prove useful in the analysis of our algorithms that use the *P-signature*.

Theorem 5.1.1 *(Fundamental Theorem of Arithmetic)[140]*
Every positive integer, except 1*, can be represented in exactly one way apart from permutations as a product of one or more primes.*

Theorem 5.1.2 *(Prime Number Theorem)[140]*
$\pi(n) \sim \frac{n}{\ln n}$*, where* $\pi(n)$ *is the number of primes less than* n*.*

Corollary 5.1.3 *[140]* $p_n \sim n \log n$*, where* p_n *is the* n_{th} *prime number.*

Theorem 5.1.4 *[12] There exists an algorithm that gives the prime numbers up to a natural number* N *in time* $O(\frac{N}{\log \log N})$*.*

Theorem 5.1.5 *[140]* $\lim_{k\to\infty} \sum_{i=1}^{k} \frac{1}{i} \to \ln(n) + \gamma$*, where* γ *is the Euler-Mascheroni constant.*

Lemma 5.1.6 *There exist an algorithm that gives the first* n *primes in time* $O(\frac{n \log n}{\log \log (n \log n)})$*.*

Proof Immediate consequence of Theorem 5.1.2 and Corollary 5.1.3. □

The following lemmas prove that the *P-signature* and the *S-signature* are abelian characteristics of a word and can be used to distinguish it from another word of the same length (in the abelian sense).

Lemma 5.1.7 *Two strings* x*,* y *of same length are represented by the same Parikh vector iff they share the same P-signature.*

Proof Immediate consequence of Theorem 5.1.1. □

Lemma 5.1.8 *Two strings x, y of same length are represented by the same Parikh vector iff they share the same S-signature.*

Proof Direct: Suppose x and y are strings of the same length and share the same *S-signature*, i.e.:

S-signature$(x) = \sum_{i=0}^{|x|-1} s(x[i]) = \sum_{i=0}^{k} a_i(n+1)^i$

S-signature$(y) = \sum_{i=0}^{|y|-1} s(y[i]) = \sum_{i=0}^{q} b_i(n+1)^i$,

where a_i is the cardinality of Σ_{i+1} in x and b_i is the cardinality of Σ_{i+1} in y.

W.l.o.g. consider $k \geq q$.

S-signature$(y) = \sum_{i=0}^{|y|-1} s(y[i]) = \sum_{i=0}^{q} b_i(n+1)^i \leq n(n+1)^q$ as $b_i \leq n$ and so *S-signature*$(y) < (n+1)^{q+1}$

Therefore $q = k$ and by using similar arguments:

$\sum_{i=0}^{q-1} b_i(n+1)^i \leq n(n+1)^{q-1} < (n+1)^q$ and so $a_k = b_q$.

Similarly it follows that $a_j = bj$ for every $j \in \{0, 1, \ldots, k\}$.

Reverse: Trivial

□

The following lemmas give us some properties of the Parikh vectors of prefixes and suffixes of a word, that simplify our computations later.

Lemma 5.1.9 *Let $rs[i]$ = minimum j such that $\mathcal{P}(y[0\,..\,i-1]) \subset \mathcal{P}(y[i\,..\,j])$, where $i \in \{1, 2, \ldots, n\}$. Then $\mathcal{P}(y[0\,..\,i-1]) \subset \mathcal{P}(y[i\,..\,q])$ for all $q \in \{rs[i], rs[i]+1, \ldots, n-1\}$ and $rs[i] \leq rs[i+1]$ for all $i \in \{1, 2, \ldots, n-1\}$.*

Proof First part:

Let $rs[i]$ = minimum j such that $\mathcal{P}(y[0\,..\,i-1]) \subset \mathcal{P}(y[i\,..\,j])$ for some $i \in \{1, 2, \ldots, n\}$.Then for $q \in \{rs[i], rs[i]+1, \ldots, n-1\}$ holds that $\mathcal{P}(y[i\,..\,q]) = \mathcal{P}(y[i\,..\,rs[i]]) + \mathcal{P}(y[rs[i]+1\,..\,q])$ and hence $\mathcal{P}(y[0\,..\,i-$

$1]) \subset \mathcal{P}(y[i\,..\,q)$.

Second part:

By definition $rs[i+1] =$ minimum j such that $\mathcal{P}(y[0\,..\,i]) \subset \mathcal{P}(y[i+1\,..\,j])$ $=$ minimum j such that $\mathcal{P}(y[0\,..\,i-1]) \subset \mathcal{P}(y[i\,..\,j]) + max(0,$ minimum k such that $(\mathcal{P}(y[i+1\,..\,i+k]) - \mathcal{P}(y[0\,..\,i]))[y[i]] > 0 - rs[i]) \geq rs[i]$.

□

Lemma 5.1.10 *Let* $re[i] =$ *maximum* j *such that* $\mathcal{P}(y[n-i\,..\,n-1]) \subset \mathcal{P}(y[j\,..\,n-i-1])$, *where* $i \in \{1, 2, \ldots, n\}$. *Then* $\mathcal{P}(y[n-i\,..\,n-1]) \subset \mathcal{P}(y[q\,..\,n-i-1])$ *for all* $q \in \{re[i], re[i]-1, \ldots, 0\}$ *and* $re[i] \geq re[i+1]$ *for all* $i \in \{1, 2, \ldots, n-1\}$.

Proof Similar to the proof of Lemma 5.1.9. □

5.1.3 The algorithms

In this section, we describe our algorithms for solving Problems 1-5. Firstly we describe some data structures that are used throughout the algorithms. Then we show how to solve the more basic problems and we extend these ideas to solve Problem 4 and Problem 5, ending with some comments on the analysis of the given algorithms.

Preprocessing

Before proceeding with the algorithms we will need some preprocessing to compute the following:

- The *S-signature* of each prefix of y is precomputed and stored in array LS, such that $\mathsf{LS}[i] = S\text{-}signature(y[0\,..\,i])$ for $0 \leq i \leq n-1$. The necessary powers of $n+1$ can be computed in $O(\sigma)$ time and stored in an array s.t. they don't have to be computed every time they are called. Then we fill the array using the properties $\mathsf{LS}[0] = s(y[0])$ and $\mathsf{LS}[i] = \mathsf{LS}[i-1] + s(y[i])$ for $1 \leq i \leq n-1$.

- The *P-signature* of each prefix of y is precomputed and stored in array Pr, such that $\mathsf{Pr}[i] = P\text{-}signature(y[0\,..\,i])$ for $0 \leq i \leq n-1$. We assume that the necessary primes can be easily found from a library in the computer. Otherwise we can produce them fast using a prime sieve as in [12] (see also Theorem 5.1.4). Then we fill the array using the properties $\mathsf{Pr}[0] = p(y[0])$ and $\mathsf{Pr}[i] = \mathsf{Pr}[i-1]p(y[i])$ for $1 \leq i \leq n-1$.

- The array rs, where $rs[i] =$ minimum j such that $\mathcal{P}(y[0\,..\,i-1]) \subset \mathcal{P}(y[i\,..\,j])$ is computed in linear time using the properties $rs[i+1] \geq rs[i]$ (Lemma 5.1.9). We use a simple sliding window approach keeping $\mathcal{P}(y[h\,..\,rs[h]]) - \mathcal{P}(y[0\,..\,h-1])$ in PV array. When trying to find $rs[h]$, if $PV[\phi[y[h-1]]] \geq 0$ then $rs[h] = rs[h-1]$, otherwise we search for $y[h-1]$ in $\{y[rs[h]\,..\,n-1\}$ and use that length as an answer, if not found we assign n to $rs[h]$ (Lemma 5.1.9).

- The array re, where $re[i] =$ maximum j such that $\mathcal{P}(y[n-i\,..\,n-1]) \subset \mathcal{P}(y[j\,..\,n-i-1])$ is computed in a similar manner to the way we compute rs so we only give the algorithm to find rs.

Preliminary problems

In this section, we describe algorithms for solving Problems 1-3. These problems are quite basic and our algorithms for Problem 4 and Problem 5 use similar ideas. The weak abelian period version of the first two problems is solved in the same manner.

Problem 1 is solved in $O(n)$ time by checking the required conditions for (h, p) to be an abelian period, i.e. the necessary Parikh vectors, using either the *S-signature* or the *P-signature* of factors of y (as of Lemmas

ALGORITHM RS(y, n,σ,ϕ)
1: $rs[0] \leftarrow 0$;
2: **for** $i \leftarrow 0$ to $\sigma - 1$ **do**
3: $PV[i] \leftarrow 0$;
4: $PV[\phi[y[h-1]]-1] \leftarrow 1$;
5: **for** $h \leftarrow 1$ to $\lfloor \frac{n-1}{2} \rfloor$ **do**
6: $PV[\phi[y[h-1]]-1] \leftarrow PV[\phi[y[h-1]]-1]-2$;
7: **if** $(PV[(\phi[y[h-1]]-1] \geq 0)$ or $(rs[h-1] = n))$ **then**
8: $rs[h] \leftarrow rs[h-1]$;
9: **else**
10: $q \leftarrow rs[h-1]$;
11: **while** $(PV[\phi[y[h-1]]-1] \neq 0)$ and $q < n-1$ **do**
12: $q \leftarrow q+1$;
13: $PV[\phi[y[q]]-1] \leftarrow PV[\phi[y[q]]-1]+1$;
14: **if** $PV[\phi[y[h-1]]-1] \neq 0$ **then**
15: $rs[h] \leftarrow n$;
16: **else**
17: $rs[h] \leftarrow q$;

5.1.7 and 5.1.8). A careful sliding window implementation would also be able to solve the problem in $O(n)$ time.

Problem 2 is solved in $O(n)$ time by the following steps:

- If $\mathcal{P}(x) \not\subset \mathcal{P}(y[0\,..\,2|x|-1])$ then the answer is immediately no.
- We calculate the array Pr or LS for rapid comparison of Parikh vectors.
- We check each (h, m) , where $0 \leq h < min(m, \lfloor \frac{n-1}{2} \rfloor + 1)$, if it is an abelian period of y until we find the first one that is. Clearly we go over at most $\lfloor \frac{n}{m} \rfloor$ factors during each period check. We check at most m different periods and hence the algorithm is linear.

Problem 3 is solved in the same way but in the last step we keep checking for abelian periods after we find the first one. Clearly we go over at most $\lfloor \frac{n}{m} \rfloor$ factors during each period check. We check m different periods and hence the linearity of the algorithm.

Identifying all weak abelian periods

This algorithm uses basic ideas from the above preliminary algorithms to solve Problem 4. Before proceeding with the algorithm the *S-signature* or the *P-signature* of each prefix of y is precomputed and stored in the array LS or the array Pr respectively. We also precompute the array re, where $re[i]$ = maximum j such that $\mathcal{P}(y[n-i\,..\,n-1]) \subset \mathcal{P}(y[j\,..\,n-i-1])$ in linear time using the properties of Lemma 5.1.10. We only show the version of the algorithm that uses the *S-signature* as it is almost the same as with the version using the *P-signature*.

ALGORITHM ALL-WEAK-ABELIAN-PERIODS-S(y, n,LS,re)
1: **for** $p \leftarrow 1$ to n **do**
2: **if** $p \geq n - re(n \bmod p) - n \bmod p$ **then**
3: $i \leftarrow 1$;
4: **while** $((i < \left\lfloor \frac{n}{p} \right\rfloor)$ and $(\mathsf{LS}[(i+1)*p-1] - \mathsf{LS}[ip-1]) = \mathsf{LS}[p-1])$ **do**
5: $i \leftarrow i+1$;
6: **if** $i = \left\lfloor \frac{n}{p} \right\rfloor$ **then**
7: Output p;

Theorem 5.1.11 *Algorithm* ALL-WEAK-ABELIAN-PERIODS-S *runs in time* $O(nlogn)$.

Proof Computation of the arrays LS and re is done in linear time as it is easy to see that each letter is checked at most once during that phase of preprocessing. During the execution of the main algorithm we go over only from some factors of y which are checked at most once. These determine the complexity of our algorithm:
$\sum_{i=1}^{|y|}$ factors of y of length i that are checked $\leq \sum_{i=1}^{|y|} \lfloor \frac{n}{i} \rfloor \leq n \sum_{i=1}^{|y|} \frac{1}{i}$
Theorem 5.1.5 states that $\lim_{k\to\infty} \sum_{i=1}^{k} \frac{1}{i} \to \ln(n) + \gamma$, where γ is the Euler-Mascheroni constant, and so we get the above result. □

Theorem 5.1.12 *Algorithm* ALL-WEAK-ABELIAN-PERIODS-S *has* $\Theta(n)$ *best case running time.*

Proof Consider an alphabet Σ. It is easy to see that the word $y = \Sigma[1]\Sigma[2]\,..\,\Sigma[\sigma]$, where $\Sigma[i]$ is the i_{th} letter of Σ, has no abelian periods. On executing our algorithms re is full of -1 and therefore we never enter the if part of the main loop of the algorithm, thus only counting from $p \leftarrow 1$ to n. No better running time is possible as preprocessing needs $\Theta(n)$ time. □

Identifying all abelian periods

We propose two algorithms for the solution of Problem 5. The first one maps each letter to a suitable number such that each factor of the string can be identified by the unique sum of the numbers corresponding to its letters (*S-signature*). The other one maps each letter to a prime number such that each factor of the string can be identified by the unique product of the numbers corresponding to its letters (*P-signature*). We are then able to perform the required checks of parikh vectors, necessary to identify abelian periods, with just one operation using ideas from algorithms from the preliminary problems.

S-Signature algorithm

This algorithm makes use of the *S-signature* of factors of y in order to make rapid comparison of Parikh vectors. It takes as input the string y, its length n and the arrays LS, rs and re and outputs all the abelian periods of y in the required encoding. For each possible h from 0 to $\lfloor \frac{n-1}{2} \rfloor$ we check all possible values of p from $rs(h) - h + 1$ to $n - h$. For (h, p) to be an abelian period we need:

1. $\mathcal{P}(y[0\,..\,h-1]) \subset \mathcal{P}(y[h\,..\,h+p-1])$,
   i.e. $p \geq rs(h) - h + 1$.

2. $\mathcal{P}(y[h\,..\,h+p-1]) = \mathcal{P}(y[h+p\,..\,h+2p-1]) = \cdots = \mathcal{P}(y[h + (((n-h) \bmod p) - 1)p\,..\,h + ((n-h) \bmod p)p - 1])$,

i.e. $(\mathsf{LS}[(i+1)*p+h-1]-\mathsf{LS}[ip+h-1]) = \mathsf{LS}[p+h-1]-\mathsf{LS}[h-1])$
for all $i \in \{2,3,\ldots,((n-h) \bmod p)-1\}$.

3. $\mathcal{P}(y[h+((n-h) \bmod p)p\,..\,n-1]) \subset \mathcal{P}(y[h\,..\,h+p-1])$,
i.e.$p \geq n - re((n-h) \mod p) - (n-h) \bmod p$.

ALGORITHM ALL-ABELIAN-PERIODS-S(y, n,LS,rs,re)
1: **for** $h \leftarrow 0$ to $\lfloor \frac{n-1}{2} \rfloor$ **do**
2: **for** $p \leftarrow rs(h)$ to $n-h$ **do**
3: **if** $p \geq n - re((n-h) \mod p) - (n-h) \bmod p$ **then**
4: $i \leftarrow 1$;
5: **while** $((i < \lfloor \frac{n-h}{p} \rfloor)$ and $(\mathsf{LS}[(i+1)*p+h-1]-\mathsf{LS}[ip+h-1]) = \mathsf{LS}[p+h-1]-\mathsf{LS}[h-1])$ **do**
6: $i \leftarrow i+1$;
7: **if** $i = \lfloor \frac{n-h}{p} \rfloor$ **then**
8: Output (h,p);

Theorem 5.1.13 *Algorithm* ALL-ABELIAN-PERIODS-S *runs in* $O(n^2)$ *time.*

Proof Computation of the arrays LS, rs and re is done in linear time as it is easy to see that each letter is checked at most once during that phase of preprocessing. The necessary powers of $n+1$ are computed in $\Theta(\sigma)$ time, where $\sigma \leq n$. During the execution of the main algorithm all the factors of y ($\frac{n(n+1)}{2}$) are checked at most once which gives time complexity $O(n^2)$. □

P-signature algorithm

This algorithm makes use of the *P-signature* of factors of y in order to make rapid comparison of Parikh vectors. It takes as input the string y, its length n and the arrays Pr, rs and re and outputs all the abelian periods of y in the required encoding. For each possible h from 0 to $\lfloor \frac{n-1}{2} \rfloor$ we check all possible values of p from $h+1$ to $n-h$. For (h,p) to be an abelian period we need:

1. $\mathcal{P}(y[0\,..\,h-1]) \subset \mathcal{P}(y[h\,..\,h+p-1])$,
   i.e. $p \geq rs(h) - h + 1$.

2. $\mathcal{P}(y[h\,..\,h+p-1]) = \mathcal{P}(y[h+p\,..\,h+2p-1]) = \cdots = \mathcal{P}(y[h + (((n-h) \bmod p) - 1)p\,..\,h + ((n-h) \bmod p)p - 1])$,
   i.e. $(\mathsf{Pr}[(i+1)*p+h-1]/\mathsf{Pr}[ip+h-1]) = \mathsf{Pr}[p+h-1]/\mathsf{Pr}[h-1])$
   for all $i \in \{1, 2, \ldots, ((n-h) \bmod p) - 1\}$

3. $\mathcal{P}(y[h + ((n-h) \bmod p)p\,..\,n-1]) \subset \mathcal{P}(y[h\,..\,h+p-1])$,
   i.e. $p \geq n - re((n-h) \bmod p) - (n-h) \bmod p$.

ALGORITHM ALL-ABELIAN-PERIODS-P(y, n,Pr)
1: **for** $h \leftarrow 0$ to $\lfloor \frac{n-1}{2} \rfloor$ **do**
2: **for** $p \leftarrow rs(h)$ to $n-h$ **do**
3: **if** $p \geq rs(h) - h + 1$ **then**
4: $i \leftarrow 1$;
5: **while** $((i < \lfloor \frac{n-h}{p} \rfloor)$ and $(\mathsf{Pr}[(i+1)*p+h-1]/\mathsf{Pr}[ip+h-1]) = \mathsf{Pr}[p+h-1]/\mathsf{Pr}[h-1])$ **do**
6: $i \leftarrow i+1$;
7: **if** $i = \lfloor \frac{n-h}{p} \rfloor$ **then**
8: Output (h, p);

Theorem 5.1.14 *Algorithm* ALL-ABELIAN-PERIODS-P *runs in* $O(n^2)$ *time.*

Proof Computation of the arrays Pr, rs and re is done in linear time as it is easy to see that each letter is checked at most once during that phase of preprocessing. During the execution of the main algorithm all the factors of y ($\frac{n(n+1)}{2}$) are checked at most once which gives time complexity $O(n^2)$. □

An example

We provide an example, providing the data structures build for the execution of our algorithm on the string $y = acabbacabbca$.

i	0	1	2	3	4	5	6	7	8	9	10	11
$y[i]$	a	c	a	b	b	a	c	a	b	b	c	a
LS$[i]$	0	13	13	14	15	15	28	28	29	30	43	43
Pr$[i]$	2	10	20	60	180	360	1800	3600	10800	32400	162000	324000
$rs[i]$	0	2	6	7	7	9	12	12	12	12	12	12
$re[i]$	11	7	6	6	3	2	-1	-1	-1	-1	-1	-1

In order to calculate the *P-signature* of factors of y we use the mapping $p : \{a, b, c\} \to \{2, 3, 5\}$, such that:
$p(a) = 2$ $p(b) = 3$ $p(c) = 5$

In order to calculate the *S-signature* of factors of y we use the mapping $s : \{a, b, c\} \to \{0, 1, 13\}$, such that:
$s(a) = 0$ $s(b) = 1$ $s(c) = 13$

All abelian periods of y are:
$(0,5)$, $(0,7)$, $(0,8)$, $(0,9)$, $(0,10)$, $(0,11)$, $(0,12)$, $(1,6)$, $(1,7)$, $(1,8)$, $(1,9)$, $(1,10)$, $(1,11)$, $(2,5)$, $(2,6)$, $(2,7)$, $(2,8)$, $(2,9)$, $(2,10)$, $(3,5)$, $(3,6)$, $(3,7)$, $(3,8)$, $(3,9)$, $(4,5)$, $(4,6)$, $(4,7)$, $(4,8)$, $(5,6)$, $(5,7)$

5.1.4 Further comments on the complexity of the above algorithms

In this subsection we give more details on the complexity of the suggested algorithms. We claim that they are optimal under the natural encoding suggested by the definition of the abelian period and that they have a best case linear running time. We also observe that a large alphabet size may lead to the creation of large numbers during the execution of our algorithms. However when dealing with applications σ is very small compared to n and so our algorithms are efficient.

Theorem 5.1.15 *Algorithm* ALL-ABELIAN-PERIODS-P *and Algorithm* ALL-ABELIAN-PERIODS-S *are optimal.*

Proof Consider the word a^n. As suggested in [114] it has $O(n^2)$ abelian periods, which is also the worst case running time of our algorithms. □

Theorem 5.1.16 *Algorithm* ALL-ABELIAN-PERIODS-P *and Algorithm* ALL-ABELIAN-PERIODS-S *have* $\Omega(n)$ *best case running time.*

Proof Consider an alphabet Σ. It is easy to see that the word $y = \Sigma[1]\Sigma[2]..\Sigma[\sigma]$, where $\Sigma[i]$ is the i_{th} letter of Σ, has no abelian periods. On executing our algorithms rs is full of n and therefore we never enter the second loop of the algorithm, thus only counting from $h \leftarrow 0$ to $\lfloor \frac{n-1}{2} \rfloor$. No better running time is possible as preprocessing needs $\Theta(n)$ time. □

As mentioned before a large alphabet size may lead to the creation of large numbers during the execution of our algorithms. In particular it is the signatures of the factors that might grow too large. The following theorems show the worst case size that they can have.

Theorem 5.1.17 *The number of digits of variables used during the execution of Algorithm* ALL-ABELIAN-PERIODS-P *is* $O(n \log(\frac{\sigma}{\log(\sigma)}))$.

Proof Consider an alphabet Σ.
The biggest variable encountered during the execution of the algorithm is the *P-signature* of the word $y = (\Sigma[\sigma|])^n$, where $\Sigma[i]$ is the i_{th} letter of Σ.
That means *P-signature*$(y) = (i_{th}$ prime number$)^n$.
As suggested by Corollary 5.1.3, *P-signature*(y) is $O((\frac{\sigma}{\log(\sigma)})^n)$. □

Theorem 5.1.18 *The number of digits of variables used during the execution of Algorithm* ALL-ABELIAN-PERIODS-S *is* $O(\sigma \log(n))$.

Proof Consider an alphabet Σ.
The biggest variable encountered during the execution of the algorithm is the *S-signature* of the word $y = (\Sigma[\sigma|])^n$, where $\Sigma[i]$ is the i_{th} letter of Σ. That means $S\text{-}signature(y) = n(n+1)^{\sigma-2}$ □

Fortunately the numbers formed when we execute Algorithm All-Abelian-Periods-P can be further reduced by taking logarithms of the signatures as shown in the definitions below:

- The *P′-signature* of a word y is defined to be equal to $\log(\prod_{i=0}^{|y|-1} p(y[i]))$.
- The array Pr' is given by $\mathsf{Pr}'[i] = \log(\prod_{j=0}^{i} p(y[j]))$

The array Pr' is useful in computing the *P′-signature* of substrings of y, as:

$$P'\text{-}signature(y[q\,..\,k]) = \begin{cases} \mathsf{Pr}[k] - \mathsf{Pr}[q-1], & q \neq 0 \\ \mathsf{Pr}[k], & q = 0 \end{cases} \tag{5.4}$$

As before the array Pr' can be easily calculated using the properties $\mathsf{Pr}'[0] = \log(p(y[0]))$ and $\mathsf{Pr}'[i] = \mathsf{Pr}'[i-1] + \log(p(y[i]))$ for $1 \leq i \leq n-1$. As of Theorem 5.1.17 the number of digits of variables used during the execution of Algorithm All-Abelian-Periods-P is $O(\log(n \log(\frac{\sigma}{\log(\sigma)}))) = O(\log n)$, while the running time of the algorithm is the same.

To conclude, the algorithms that use the *S-signature*, seem to be faster than the algorithms that use the *P-signature*. This is a consequence of the last two theorems, where it is shown that in the same problem the algorithms that use the *S-signature* are usually smaller than their corresponding *P-signatures*. Furthermore in order to compute *S-signatures* we need to perform additions and subtractions instead of multiplications and

divisions, which are required to compute *P-signatures* and are generally more time consuming.

5.2 Abelian borders in words

5.2.1 Properties

In this section we prove some properties regarding the abelian borders of a word. We also quote some results from the literature that are relevant to the structure of Thue-Morse words.
The following lemma shows that the abelian borders of a word x occur in complementary pairs, i.e. pairs whose lengths sum up to n.

Lemma 5.2.1 *The abelian borders of a word x occur in complementary pairs, i.e. pairs whose lengths sum up to n, except the abelian border with length $\frac{n}{2}$, if it exists.*

Proof Direct consequence of the definition of the abelian border. □

The following lemma shows that the shortest abelian border and the longest abelian border of a word are also complementary, i.e. the sum of their lengths is equal to the length of the word.

Lemma 5.2.2 *The length of the longest abelian border of a word $x[1\,..\,n]$, b, is equal to $n - b'$, where b' denotes the length of the shortest abelian border of x.*

Proof Let $x[1\,..\,b]$, where $1 \leq b < n$, be the longest abelian border of x. By definition $x[b+1\,..\,n]$ is an abelian border of x.
Suppose there exists a shorter abelian border $x[j\,..\,n]$, where $b+1 < j \leq n$. Then $x[1\,..\,j-1]$ is an abelian border longer than $x[1\,..\,b]$ (contradiction). □

Corollary 5.2.3 *The longest abelian border of a word $x[1..n]$ has length greater than or equal to $\lceil n/2 \rceil$, if it exists.*

The following lemmas provide us with information about the structure of Thue-Morse words.

Lemma 5.2.4 *[25] T_{2n} is a palindrome for any $n > 1$.*

5.2.2 Identifying all abelian borders

In this section we will show a $\Theta(n)$ algorithm for the identification of all abelian borders of a word. However, before proceeding with the algorithm we need to introduce the vector $V(i)$, which gives the difference between the Parikh vectors of the prefix and the suffix of x of length i, i.e.

$$V(i) = \mathcal{P}(x[1\,..\,i]) - \mathcal{P}(x[n-i+1\,..\,n]).$$

Algorithm All-Abelian-Borders computes the vector $V(i)$ introduced before for $1 \leq i < \lfloor \frac{n}{2} \rfloor$ and outputs i (the length of a prefix/suffix which is an abelian border of x) if $V = 0$. Due to complementarity of the abelian borders (Lemma 5.2.1) the length of the rest of the prefixes that are abelian borders of x is given by

$$n - \text{the length of an abelian border found before.}$$

This algorithm implements ideas presented in the algorithm in [?], which outputs all the abelian squares in x. An abelian border of a word x is simply an abelian square in xx having its centre in the centre of xx. The algorithm of Cummings and Smyth in [?] can identify such squares in linear time. Similar algorithms can also be designed using the P-signature or the S-signature of factors of x (as defined in [60]).

ALGORITHM ALL-ABELIAN-BORDERS(x, n, σ, f)

```
1: V ← 0;
2: zeros ← σ;
3: V[f(x[1])] ← V[f(x[1])] + 1;
4: V[f(x[n])] ← V[f(x[n])] − 1;
5: if V[f(x[n])] = 0 then
6:    Output 1;
7: else
8:    zeros ← σ − 2;
9: for i ← 2 to ⌊n/2⌋ do
10:   V[f(x[i])] ← V[f(x[i])] + 1;
11:   if V[f(x[i])] = 0 then
12:      zeros ← zeros + 1;
13:   if V[f(x[i])] = 1 then
14:      zeros ← zeros − 1;
15:   V[f(x[n − i + 1])] ← V[f(x[n − i + 1])] − 1;
16:   if V[f(x[n − i + 1])] = 0 then
17:      zeros ← zeros − 1;
18:   if V[f(x[n − i + 1])] = −1 then
19:      zeros ← zeros + 1;
20:   if zeros = σ then
21:      Output i;
```

5.2.3 Abelian borders in Thue-Morse words

In this section we investigate the appearance of abelian borders in Thue-Morse words. More specifically we identify all the abelian borders of Thue-Morse words, using their definition via morphism.

Theorem 5.2.5

$$\textit{All abelian borders of } T_n \textit{ are:} \begin{cases} \textit{all prefixes of } T_n - \{\emptyset, T_n\} \\ , if n \textit{ is even} \\ \textit{all prefixes of } T_n \textit{ of even length} \\ -\{\emptyset, T_n\}, if n \textit{ is odd} \end{cases}$$

Proof The first case is a direct consequence of Lemma 5.2.4. In the case that n is odd we will use the definition of a Thue-Morse word via morphisms. Let $a(x)$ be a function giving the number of zeros that exist

in a word x. Then:

$$\begin{aligned} a(T_n[1\,..\,2k]) &= a(m(T_{n-1}[1])m(T_{n-1}[2])\ldots m(T_{n-1}[k])) \\ &= a(m(T_{n-1}[|T_{n-1}|-k+1])m(T_{n-1}[|T_{n-1}|-k+2]) \\ &\ldots m(T_{n-1}[|T_{n-1}|])])) \\ &= a(T_n[|T_n|-2k+1\,..\,|T_n|])) \end{aligned}$$

Therefore all non-empty prefixes of T_n of even length are abelian borders of T_n. In the case of a prefix of odd length the Parikh vectors of the prefix minus its last letter and the suffix minus its first letter match (as shown above). Lemma 5.2.4 suggests that the last letter of the prefix and the first letter of the suffix result from the action of the morphism on the same letter of T_{n-1}, thus producing a different letter. □

5.2.4 Average case analysis

In this section we investigate properties of the abelian borders of a word in the average case. More specifically we comment on the average length of the abelian borders of a word and the frequency of abelian borders in a binary word in the average case.

The average length of the abelian borders of a word is found using the complementarity property of abelian borders as shown in the theorem below.

Theorem 5.2.6 *The average length of the abelian borders of a word x of length n, in the case that x has abelian borders, is $\frac{n}{2}$.*

Proof Obviously if x has no abelian borders then the average length of its abelian borders(ℓ) does not exist. In the case that x has abelian

borders Lemma 5.2.1 holds and therefore:

$$\begin{aligned}\ell &= \frac{\text{Sum of the lengths of abelian borders of } x}{\text{number of abelian borders of } x} \\ &= \frac{(\text{number of abelian borders of } x)\frac{n}{2}}{\text{number of abelian borders of } x} = \frac{n}{2}\end{aligned}$$

□

It seems that abelian borders appear frequently in a word. In order to comment on that frequency we will study the behaviour of the average number of abelian borders in W_n, the set of binary words of length n. This average is the same as the expected value of abelian borders when we consider a binary word x, $|x| = n$ with the letters of the word drawn independently from (a, b) with a constant probability distribution $(\frac{1}{2}, \frac{1}{2})$. Bounds on the expected number of abelian borders of a binary word x of length n are given below using combinatorial identities for central binomial coefficients as well as simple calculus.

Lemma 5.2.7 *A binary word of length n has $\Omega(\sqrt{n})$ abelian borders on average.*

Proof Let x be a binary word of length n, with its letters drawn independently from (a, b) with a constant probability distribution $(\frac{1}{2}, \frac{1}{2})$.

$$\text{Let } X_i = \begin{cases} 0, & x[1\,..\,i] \text{ is not an abelian border of } x \\ 1, & x[1\,..\,i] \text{ is an abelian border of } x \end{cases}$$

The expected number of abelian borders of x, where n is odd, is:

$$\begin{aligned}
E(\sum_{i=1}^{n} X_i) &= \sum_{i=1}^{n} E(X_i) \text{ (linearity of operator)} \\
&= 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} E(X_i) \text{ (Lemma 5.2.1)} \\
&= 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (\text{number of binary words of length } n \text{ with } x[1\,..\,i] \\
&\quad \text{as an abelian border})/|W_n| \\
&= 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{(\sum_{j=0}^{i} \binom{i}{j}^2) 2^{n-2i}}{2^n} = 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\binom{2i}{i}}{2^{2i}} \text{ (Lemma 3.1.1)} \\
&\geq \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\sqrt{i}} \text{ (Lemma 3.1.2)} \\
&\geq \int_{1}^{\lfloor \frac{n}{2} \rfloor + 1} \frac{1}{\sqrt{x}}\,\mathrm{d}x \text{ (in Figure 5.1 the area of the shaded} \\
&\quad \text{rectangles is greater than the area between the curve} \\
&\quad y = \frac{1}{\sqrt{x}} \text{ and the x-axis)} \\
&= \left[2\sqrt{x}\right]_1^{\lfloor \frac{n}{2} \rfloor + 1} = 2\sqrt{\lfloor \frac{n}{2} \rfloor + 1} - 2 = \sqrt{2(n+1)} - 2
\end{aligned}$$

Similarly when n is even we get:

$$\begin{aligned}
E(\sum_{i=1}^{n} X_i) &\geq \sqrt{2n} - 2 + E(X_{\frac{n}{2}}) = \sqrt{2n} - 2 + \frac{\binom{n}{\frac{n}{2}}}{2^n} \\
&\geq \sqrt{2n} - 2 + \frac{1}{2\sqrt{\frac{n}{2}}} \text{ (Lemma 3.1.2)} \\
&= \sqrt{2n} - 2 + \frac{1}{\sqrt{2n}}
\end{aligned}$$

□

Lemma 5.2.8 *A binary word x of length n has $O(\sqrt{n})$ abelian borders on average.*

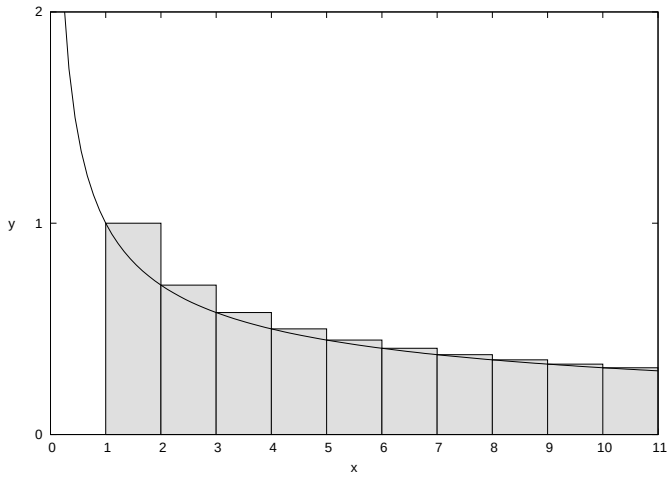


Figure 5.1: Geometrical interpretation of the inequality $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} \geq \int_1^{n+1} \frac{1}{\sqrt{x}}\,\mathrm{d}x$ for $n = 10$

Proof Lemma 5.2.7 suggests that the expected number of abelian borders of x, when n is odd, is:

$$
\begin{aligned}
E(\sum_{i=1}^{n} X_i) &= 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\binom{2i}{i}}{2^{2i}} \leq 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\frac{4^i}{\sqrt{2i}}}{2^{2i}} \text{ (Lemma 3.1.2)} \\
&= 2\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\sqrt{2i}} \\
&\leq \sqrt{2} \int_0^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\sqrt{x}}\,\mathrm{d}x \text{ (in Figure 5.2 the area of the shaded} \\
&\quad \text{rectangles is smaller than the area between the curve} \\
&\quad y = \frac{1}{\sqrt{x}} \text{ and the x-axis.)} \\
&= \sqrt{2}\left[2\sqrt{x}\right]_0^{\lfloor \frac{n}{2} \rfloor} = 2\sqrt{n-1}
\end{aligned}
$$

Similarly when n is even we get:

$$
\begin{aligned}
E(\sum_{i=1}^{n} X_i) &\leq 2\sqrt{2(\frac{n}{2}-1)} + E(X_{\frac{n}{2}}) \leq 2\sqrt{n-2} + \frac{1}{\sqrt{2 \cdot \frac{n}{2}}} \text{ (Lemma 3.1.2)} \\
&= 2\sqrt{n-2} + \frac{1}{\sqrt{n}}
\end{aligned}
$$

□

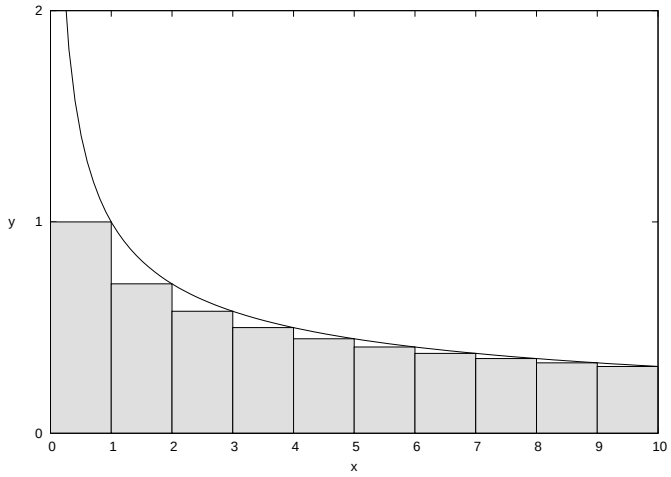


Figure 5.2: Geometrical interpretation of the inequality $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} \leq \int_0^n \frac{1}{\sqrt{x}}\,\mathrm{d}x$ for $n = 10$

We summarise the above results in the following theorem.

Theorem 5.2.9 *A binary word x of length n has $\Theta(\sqrt{n})$ abelian borders on average. In fact if this quantity is denoted by $EAB(x)$ we have:*

$$\sqrt{2(n+1)} - 2 \leq EAB(x) \leq 2\sqrt{n-1}, \text{ if } n \text{ is odd}$$

or

$$\sqrt{2n} - 2 + \frac{1}{\sqrt{2n}} \leq EAB(x) \leq 2\sqrt{n-2} + \frac{1}{\sqrt{n}}, \text{ if } n \text{ is even.}$$

5.2.5 Abelian borders in binary words

Before proceeding we need to introduce some more definitions. A *Dyck* word of length $2n$ is a binary string consisting of n zeros and n ones such that no prefix of the string has more ones than zeros. It is known that Catalan numbers enumerate Dyck words ([144]). The n^{th} *Catalan number* is given in terms of binomial coefficients:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \qquad \text{for } n \geq 0$$

Let W_n denote the set of binary words of length n, and S_n denote the subset of W_n having no abelian borders. The first elements of S_n are:

$$S_1 = \{0, 1\}, \quad S_2 = \{01, 10\}, \quad S_3 = \{001, 011, 100, 110\},$$

$$S_4 = \{0001, 0011, 0111, 1000, 1100, 1110\}$$

Similarly, we denote by $S_n^{'}$ the complementary set of S_n, the set of binary words of length n having at least one abelian border. Its first elements are:

$$S_2^{'} = \{00, 11\}, \quad S_3^{'} = \{000, 010, 101, 111\},$$

$$S_4^{'} = \{0000, 0010, 0100, 0110, 1001, 1011, 1101, 1111, 0101, 1010\}$$

In the following lemma, we establish the relation of abelian borders to Dyck words.

Lemma 5.2.10 *A binary word x of length n has a shortest abelian border of length k, $2 \leq k \leq \lfloor \frac{n}{2} \rfloor$, iff $x[1\,..\,k]$ is the shortest prefix of x that contains a Dyck word (or its bitwise negation) of length $0 < 2h \leq k$ as a subsequence.*

Proof

(Only if case)

Let y be a word of length k, with its letters defined as follows:

$$y[i] = \begin{cases} a, & \text{if } x[i] = x[n+1-i] \\ b, & \text{if } x[i] = 0 \text{ and } x[n+1-i] = 1 \\ c, & \text{if } x[i] = 1 \text{ and } x[n+1-i] = 0 \end{cases}$$

Let z be a subsequence of y of length ℓ, constructed by removing all a's from y. If z begins with a b, then every prefix $z[1\,..\,j]$ of z, where $1 \leq j \leq \ell - 1$, contains more b's than c's. But, since x has an abelian border of length k, the number of b's in z is the same as the number of c's. Therefore, z is a Dyck word (with b corresponding to 0 and c corresponding to 1). When z begins with a c the situation is similar.

(If case)

Following the reverse procedure, we can construct every word having a shortest abelian border of length k by finding the appropriate Dyck word and interspersing its symbols with zeros and ones, as shown above. □

In [144] it was established that the number of Dyck words of length $2n$ is the n^{th} Catalan number, C_n. Catalan numbers are bounded as follows [32]:

$$\frac{2^{2n}}{n+1} \cdot \sqrt{\frac{1}{\pi n}(1 - \frac{1}{4n})} < C_n < \frac{2^{2n}}{(n+1)\sqrt{\pi n}} \tag{5.5}$$

The following lemmas provide bounds on the number of words in W_n that have shortest abelian border of length k.

Lemma 5.2.11 *The number of binary words of length n with a shortest abelian border of length k is $O(\frac{2^n}{k\sqrt{k}})$. In fact that number is at most $2\sqrt{2} \cdot \frac{2^n}{k\sqrt{\pi k}} + o(\frac{2^n}{k\sqrt{\pi k}})$.*

Proof Clearly, if $k = 1$ the binary words of length n with shortest abelian border of length 1 are of form $0x0$ or $1x1$ where $x \in W_{n-1}$. The number of these words is 2^{n-1}, which verifies the above statement.

For $k \geq 2$, Lemma 5.2.10 suggests that the shortest border $x[1\,..\,k]$ contains a Dyck word (or its binary negation) as a subsequence. Therefore, the number of binary words of length n with shortest abelian border of length k is:

$$
\begin{aligned}
&2\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} (\text{number of Dyck words of length } 2i)\times \\
&\qquad (\text{number of ways for their placement})\times \\
&\qquad\quad (\text{words for the rest of the positions of the borders})\times \\
&\qquad\quad (\text{subwords for the rest of the word}) \\
=&2\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} C_i \cdot \binom{k-2}{2i-2} \cdot 2^{k-2i} \cdot 2^{n-2k} \qquad (\text{see } [144]) \\
\leq&2\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{2^{2i}}{(i+1)\sqrt{\pi i}} \binom{k-2}{2i-2} \cdot 2^{n-k-2i} \qquad (\text{see eq. } (5.5)) \\
=&\frac{2^{n+1-k}}{\sqrt{\pi}} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{1}{(i+1)\sqrt{i}} \binom{k-2}{2i-2} \\
\leq&\frac{8 \cdot 2^{n-k}}{\sqrt{\pi}} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{\sqrt{i}}{2i(2i-1)} \binom{k-2}{2i-2} \text{ (We try to form } \binom{k}{2i}) \\
\leq&\frac{8 \cdot 2^{n-k}}{\sqrt{\pi}k(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \sqrt{i} \binom{k}{2i} \leq \frac{8 \cdot 2^{n-k}}{\sqrt{\pi}k(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \sqrt{\frac{k}{2}} \binom{k}{2i} \\
=&\frac{4\sqrt{2} \cdot 2^{n-k}}{\sqrt{k\pi}(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{2i} \leq \frac{4\sqrt{2} \cdot 2^{n-k}}{\sqrt{k\pi}(k-1)} \cdot 2^{k-1} \text{ (as } \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k} = 2^{n-1} \text{ [23])} \\
=&\frac{2\sqrt{2} \cdot 2^{n}}{(k-1)\sqrt{\pi k}}
\end{aligned}
$$

□

Lemma 5.2.12 *The number of binary words of length n with a shortest abelian border of length k is* $\Omega(\frac{2^n}{k\sqrt{k}})$. *In fact that number is at least*

$\sqrt{2} \cdot \frac{2^n}{k\sqrt{\pi k}} + o(\frac{2^n}{k\sqrt{\pi k}})$.

Proof The case for $k = 1$ is covered in Lemma 5.2.11.

For $k > 2$, proceeding as in Lemma 5.2.11, we get:

$$
\begin{aligned}
&2\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} C_i \binom{k-2}{2i-2} \cdot 2^{n-k-2i} \\
&\geq 2^{n+1-k} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{2^{2i}}{(i+1)} \sqrt{\frac{1}{\pi i}(1-\frac{1}{4i})} \binom{k-2}{2i-2} \cdot 2^{-2i} \qquad \text{(see eq. (5.5))} \\
&= \frac{2^{n+1-k}}{\sqrt{\pi}} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{\sqrt{4i-1}}{2i(i+1)} \binom{k-2}{2i-2} \\
&= \frac{2^{n+1-k}}{\sqrt{\pi}} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{\sqrt{4i-1}}{2i(i+1)} \cdot \frac{2i(2i-1)}{k(k-1)} \binom{k}{2i} \\
&= \frac{2^{n+1-k}}{\sqrt{\pi}k(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{(2i-1)\sqrt{4i-1}}{i+1} \binom{k}{2i} \\
&\geq \frac{2^{n+1-k}}{\sqrt{\pi}k(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{(2i-1)(\sqrt{4i}-1)}{i+1} \binom{k}{2i} \text{ (as } \sqrt{4i-1} \geq \sqrt{4i}-1) \\
&= \frac{2^{n+1-k}}{\sqrt{\pi}k(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} (2\sqrt{4i} - 2 - \frac{3(\sqrt{4i}-1)}{i+1}) \binom{k}{2i} \\
&\geq \frac{2^{n+1-k}}{\sqrt{\pi}k(k-1)} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} (2\sqrt{4i} - 5) \binom{k}{2i} \text{ (as } \frac{\sqrt{4i}-1}{i+1} < 1 \text{ for } i > 0) \\
&\geq \frac{2^{n+1-k}}{\sqrt{\pi}k(k-1)} (-5 \cdot 2^{k-1} + 2\sqrt{2} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} \frac{2i}{\sqrt{k}} \binom{k}{2i}) \\
&\text{(as } \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k} = 2^{n-1} \text{ [23])} \\
&\geq \frac{2^{n+1-k}}{\sqrt{\pi}k(k-1)} (-5 \cdot 2^{k-1} + 2\sqrt{2k} \cdot 2^{k-2}) \quad (\text{as } \sum_{k=0}^{n} k \binom{n}{k} = n2^{n-1} \text{ [23]})
\end{aligned}
$$

□

We summarize the above results in the following theorem.

Theorem 5.2.13 *The number of binary words of length n with shortest abelian border of length k is $\Theta(\frac{2^n}{k\sqrt{k}})$. In fact, that number is $2\sqrt{2}\cdot\frac{2^n}{k\sqrt{\pi k}} + o(\frac{2^n}{k\sqrt{\pi k}})$.*

One can apply the above results directly to get bounds on the size of S_n or S_n'. However, this would yield very broad bounds. In order to get tighter bounds, we employ reduction techniques as shown in the following propositions.

Proposition 5.2.14 *$|S_n'|$ is $\Theta(2^n)$. In fact, $|S_n'|$ lies between $\frac{2}{3}\cdot 2^n - \frac{2}{3}$ and 2^n, when n is even, and between $\frac{1}{3}\cdot 2^n - \frac{2}{3}$ and 2^n, when n is odd.*

Proof The upper bound is obvious. For the lower bound, we will consider how to construct the entries that belong to S_n':

- $0x0$ and $1x1$, where x is a binary word with length $n-2$.
- $01x01$ and $10x10$, where x is a binary word with length $n-4$.
- $001x001$ and $110x110$, where x is a binary word with length $n-6$.
- ...
- $x1x1$ and $y0y0$, where x is a word composed by $\frac{n}{2}-1$ zeros and y is a word composed by $\frac{n}{2}-1$ ones.

That gives:

$$\begin{aligned} S_n' &\geq 2\cdot 2^{n-2} + 2\cdot 2^{n-4} + 2\cdot 2^{n-6} + \cdots + 2\cdot 2^{n-2} + 2\cdot 2^2 + 2\cdot 2^0 \\ &= 2(2^{n-2} + 2^{n-4} + 2^{n-6} + \cdots + 2^{n-2} + 2^2 + 1) \\ &= 2\frac{4^{\frac{n-2}{2}+1} - 1}{4-1} = \frac{2}{3}(2^n - 1) \end{aligned}$$

Similarly, when n is odd we get that $|S_n'| \geq \frac{1}{3}\cdot 2^n - \frac{2}{3}$. □

Proposition 5.2.15 *$|S_n|$ is $\Omega((\frac{3+\sqrt{13}}{2})^{\frac{n}{2}})$.*

Proof First, consider that n is even with $n = 2k$, $k \geq 3$ and $x \in S_n$. Then, x can be written as the concatenation $x = x_1x_2x_3$, where $x_1x_3 \in S_{n-2}$, $|x_1| = |x_3| = k-1$, and $|x_2| = 2$. Obviously, if $x_1[1] = 0$ then $x_3[k-1] = 1$ and $x_2 \in \{00, 01, 11\}$ always gives a valid case. On the other hand, if $x_1[1] = 1$ then $x_3[k-1] = 0$ and $x_2 \in \{00, 10, 11\}$ always gives a valid case. Therefore, $S_n \geq 3S_{n-2}$.

Similarly, x can be written as $x = x_1x_2x_3$ where $x_1x_3 \in S_{n-4}$ and $|x_2| = 4$. In this case, if $x_1[1] = 0$ then $x_3[k-1] = 1$ and $x_2 = 0101$ always gives a valid case. On the other hand, if $x_1[1] = 1$ then $x_3[k-1] = 0$ and $x_2 = 1010$ always gives a valid case. Therefore $|S_n| \geq 3|S_{n-2}| + |S_{n-4}|$.

Solving the above recurrence relation with initial conditions $|S_2| = 2$ and $|S_4| = 6$, yields the solution $-\frac{2}{\sqrt{13}}(\frac{3-\sqrt{13}}{2})^{\frac{n}{2}} + \frac{2}{\sqrt{13}}(\frac{3+\sqrt{13}}{2})^{\frac{n}{2}}$, which completes the proof.

In the case that n is odd, $n = 2k+1$ with $k \geq 3$ and $x \in S_n$, we have $x = x_1x_2x_3$ where $x_1x_3 \in S_{n-1}$, $|x_2| = |x_3| = k$ and $x_2 \in \{0, 1\}$. Therefore:

$$\begin{aligned} S_n = 2S_{n-1} &\geq 2 \cdot (-\frac{2}{\sqrt{13}}(\frac{3-\sqrt{13}}{2})^{\frac{n-1}{2}} + \frac{2}{\sqrt{13}}(\frac{3+\sqrt{13}}{2})^{\frac{n-1}{2}}) \\ &= -\frac{4}{\sqrt{13}}(\frac{3-\sqrt{13}}{2})^{-\frac{1}{2}}(\frac{3-\sqrt{13}}{2})^{\frac{n}{2}} + \frac{4}{\sqrt{13}}(\frac{3+\sqrt{13}}{2})^{-\frac{1}{2}}(\frac{3+\sqrt{13}}{2})^{\frac{n}{2}} \end{aligned}$$

□

By using Theorem 5.2.13 we obtain bounds for the average size of the shortest border of words in S'_n, as shown below.

Theorem 5.2.16 *On average a word in S'_n has shortest border of length $\Theta(\sqrt{n})$. In fact that number lies between $\frac{2\sqrt{2}}{\sqrt{\pi}}(\sqrt{2(n+2)} - 2)$ and $\frac{6n}{\sqrt{\pi}}$.*

Proof As in Theorem 5.2.13, on average a word in S'_n has shortest border

of length:

$$\ell = \frac{\sum_{k=1}^{\frac{n}{2}} k \cdot (\text{binary words with shortest abelian border of length } k)}{|S'_n|}$$
$$= \frac{\sum_{k=1}^{\frac{n}{2}} 2\sqrt{2} \cdot \frac{2^n}{\sqrt{\pi k}} + o(\frac{2^n}{\sqrt{\pi k}})}{|S'_n|}$$

Proposition 5.2.14 shows that:

$$\sum_{k=1}^{\frac{n}{2}} 2\sqrt{2} \cdot \frac{1}{\sqrt{\pi k}} + o(\frac{1}{\sqrt{\pi k}}) \leq \ell \leq 3(\sum_{k=1}^{\frac{n}{2}} 2\sqrt{2} \cdot \frac{1}{\sqrt{\pi k}} + o(\frac{1}{\sqrt{\pi k}}))$$

As shown in Figure 5.3, we can bound the sum $\sum_{k=1}^{\frac{n}{2}} \frac{1}{\sqrt{k}}$ as follows:

$$\begin{aligned}
&\int_1^{\lfloor \frac{n}{2} \rfloor + 1} \frac{1}{\sqrt{x}}\,\mathrm{d}x \quad \leq \sum_{k=1}^{\frac{n}{2}} \frac{1}{\sqrt{k}} \leq \int_0^{\lfloor \frac{n}{2} \rfloor} \frac{1}{\sqrt{x}}\,\mathrm{d}x \\
\Leftrightarrow\ &\left[2\sqrt{x}\right]_1^{\lfloor \frac{n}{2} \rfloor + 1} \quad \leq \sum_{k=1}^{\frac{n}{2}} \frac{1}{\sqrt{k}} \leq \left[2\sqrt{x}\right]_0^{\lfloor \frac{n}{2} \rfloor} \\
\Leftrightarrow\ &\sqrt{2(n+2)} - 2 \leq \sum_{k=1}^{\frac{n}{2}} \frac{1}{\sqrt{k}} \leq \sqrt{2n}
\end{aligned}$$

Therefore:

$$\frac{2\sqrt{2}}{\sqrt{\pi}}(\sqrt{2(n+2)} - 2) + o(\sqrt{n}) \leq \ell \leq \frac{12\sqrt{n}}{\sqrt{\pi}} + o(\sqrt{n})$$

□

5.2.6 Identifying the shortest abelian border

The results shown in the previous section have applications to the analysis of algorithms identifying abelian borders. More specifically, we give an algorithm that finds the shortest abelian border of a word, thus identifying whether a word is abelian border-free, and we prove that it has $\Theta(\sqrt{n})$ average running time in the case that our word belongs in S'_n.

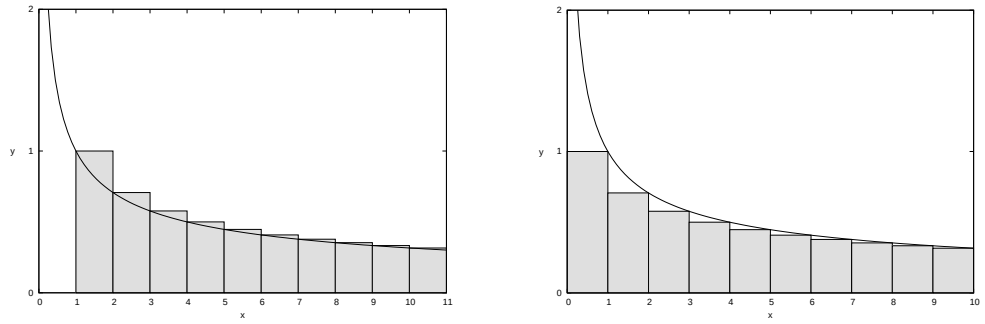


Figure 5.3: Geometrical interpretation of the inequality $\int_1^{n+1} \frac{1}{\sqrt{x}}\,\mathrm{d}x \leq \sum_{i=1}^{n} \frac{1}{\sqrt{i}} \leq \int_0^n \frac{1}{\sqrt{x}}\,\mathrm{d}x$ for $n = 10$

Before proceeding with the algorithm, we need to introduce the vector $V_x(i)$, which gives the difference between the Parikh vectors of the prefix and the suffix of x of length i, i.e.

$$V_x(i) = \mathcal{P}(x[1\,..\,i]) - \mathcal{P}(x[n-i+1\,..\,n]).$$

ALGORITHM SHORTEST-ABELIAN-BORDER(x, n, σ, ϕ)

```
 1: V ← 0;
 2: zeros = σ;
 3: V[φ(x[1])] ← V[φ(x[1])] + 1;
 4: V[φ(x[n])] ← V[φ(x[n])] − 1;
 5: if V[φ(x[n])] = 0 then
 6:    Output 1 and HALT;
 7: else
 8:    zeros = σ − 2;
 9: for i ← 2 to ⌊n/2⌋ do
10:    V[φ(x[i])] ← V[φ(y[i])] + 1;
11:    if V[φ(x[i])] = 0 then
12:       zeros = zeros + 1;
13:    if V[φ(x[i])] = 1 then
14:       zeros = zeros − 1;
15:    V[φ(x[n − i + 1])] ← V[φ(x[n − i + 1])] − 1;
16:    if V[φ(x[n − i + 1])] = 0 then
17:       zeros = zeros + 1;
18:    if V[φ(x[n − i + 1])] = −1 then
19:       zeros = zeros + 1;
20:    if zeros = σ then
21:       Output i and HALT;
22: Output n and HALT;
```

Algorithm SHORTEST-ABELIAN-BORDER computes the vector V_x and outputs i (the length of a prefix/suffix which is an abelian border of x) whenever $V = 0$. The abelian borders of a word x occur in complementary pairs, i.e. pairs whose lengths sum to n, except the abelian border with length $\frac{n}{2}$, if it exists [49]. Due to this complementarity of the abelian borders we only need to check prefixes of x with length at most $\lfloor \frac{n}{2} \rfloor$.

It is easy to observe that the algorithm works in $O(n)$ time. In the next lines we will prove that in the case that a string from S'_n is given as input the algorithm works in $\Theta(\sqrt{n})$ time on average.

Theorem 5.2.17 *Algorithm* SHORTEST-ABELIAN-BORDER *computes the shortest abelian border of a string in* S'_n *in* $\Theta(\sqrt{n})$ *time on average.*

Proof Clearly, the running time of Algorithm SHORTEST-ABELIAN-BORDER is proportional to the length of the shortest border of x, which is $\Theta(\sqrt{n})$ on average by Theorem 5.2.16. □

Chapter 6

Fibonacci words

Fibonacci words are important in many concepts [24] and are often cited as a worst case example for many string algorithms. Over the years much research has been done on them, e.g. locating all factors of a Fibonacci string [69], characterizing all squares of a Fibonacci string [118, 151], identifying all runs of a Fibonacci string[167, 169], locating all maximal quasiperiodicities of a Fibonacci string [133], identifying all covers of a circular Fibonacci string [152], identifying all borders of a Fibonacci string [85], finding palindromes of a Fibonacci string [97], etc. Some research has also been extended to Tribonacci strings [209, 211, 228].

Unfortunately not much is known about quasiperiodicities in Fibonacci words. We prove some properties of seeds, covers, periods and borders used later for finding quasiperiodicities in Fibonacci strings. We are then able to identify quasiperiodicities in Fibonacci strings (Section 6.1.2) and circular Fibonacci strings (Section 6.1.3). We give further comments on the number of distinct seeds in Fibonacci strings and [62, 63] we investigate the appearance of overlapping factors in Fibonacci words which will help to provide some bounds on the maximum number of distinct overlapping factors in a word [52]. Furthermore we comment on the appearance of abelian borders in Fibonacci words [49].

After a series of efforts in the last years ([81, 82, 123, 129, 168, 192, 191, 202, 214, 215]) modified Padovan words have provided the current lower bound for the maximum number of runs in a word [220]. However Simpson [220] altered the definition of the Padovan words using some reversing functions to produce run richer words. In this chapter, we give the first formal study of the Padovan word. We formally define it similarly to [220] and we identify all its borders and covers as well as revealing some interesting properties regarding its factor structure, its squares and its cubes using techniques similar to the ones in [62, 63, 52].

6.1 Quasiperiodicities in Fibonacci strings

6.1.1 Properties

In this section, we quote from the literature some properties for the borders and the factor structure of Fibonacci words that will prove useful later on. We also prove some additional facts about the factor structure of Fibonacci words.

Lemma 6.1.1 *[223] The letters a of F_∞ can be found at the positions given by the successive values of the Lower Wythoff sequence (OEIS A000201): $\lfloor n\phi \rfloor$*

Lemma 6.1.2 *[223] The letters b of F_∞ can be found at the positions given by the successive values of the Upper Wythoff sequence (OEIS A001950): $\lfloor n\phi^2 \rfloor$*

Lemma 6.1.3 *[85]*

$$\textit{All borders of } F_n \textit{ are:} \begin{cases} \{\}, & n \in \{0,1,2\} \\ \{F_{n-2}, F_{n-4}, F_{n-6}, \dots F_1\}, & n = 2k+1, k \geq 1 \\ \{F_{n-2}, F_{n-4}, F_{n-6}, \dots F_2\}, & n = 2k, k \geq 2 \end{cases} \tag{6.1}$$

Lemma 6.1.4 *[85] For $n \geq 1$, the set of nonâĂŞempty borders of F_n is*

$$\{F_n, F_{n-2}, F_{n-4}, F_{n-6}, \dots F_k\}$$

where $k = 1$ if n is odd and $k = 2$ if n is even.

Lemma 6.1.5 *[151] $F_k = P_k \delta_k$, where $P_k = F_{k-2} F_{k-3} .. F_1$, $k \geq 2$ and $\delta_k = ab$ if k is even or $\delta_k = ba$ otherwise.*

Proof Easily proved by induction. □

It is sometimes useful to consider the expansion of a Fibonacci string as a concatenation of two Fibonacci factors. We define the (F_m, F_{m-1})-expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-1\}$, as follows:

- Expand F_n using the recurrence formula as $F_{n-1} F_{n-2}$.
- Expand F_{n-1} using the recurrence formula as $F_{n-2} F_{n-3}$.
- Keep expanding as above until F_{m+1} is expanded.

Lemma 6.1.6 *The (F_m, F_{m-1})-expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-1\}$ is unique.*

Proof Easily proved by induction. □

Lemma 6.1.7 *The starting positions of the occurrences of F_m in F_n are the starting positions of the factors considered in the (F_m, F_{m-1})-expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-1\}$ except from the last F_{m-1}, if it is a border of F_n.*

Proof

Using the recurrence relation we can get the (F_m, F_{m-1})-expansion of F_n as shown before:

$F_n = F_m F_{m-1} F_m F_m F_{m-1} F_m F_{m-1} F_m \dots$

We can now observe many occurrences of F_m in F_n. Any other occurrence should have one of the following forms (note that there are no consecutive F_{m-1} in the above expansion):

- xy, where x is a nonempty suffix of F_m and y a nonempty prefix of F_{m-1}. Then both x and y are borders of F_m. It holds that $|x| + |y| = |F_m| = |F_{m-1}| + |F_{m-2}|$, but $|x| \leq |F_{m-2}|$, $|y| \leq |F_{m-2}|$ and so there exists no such occurrence of F_m in F_n.

- xy, where x is a nonempty suffix of F_{m-1} and y a nonempty prefix of F_m. Then y is also a border of F_m and so belongs to $\{F_{m-2}, F_{m-4}, \dots F_1\}$, if n is odd, or to $\{F_{m-2}, F_{m-4} \dots F_2\}$, otherwise. But $|x| + |y| = |F_m|$ and $0 < |x| \leq |F_{m-1}|$ so in either case the only solution is $x = F_{m-1}$ and $y = F_{m-2}$ giving the occurrences of F_m at the starting positions of F_{m-1} in the above expansion.

- $xF_{m-1}y$, where x is a nonempty suffix of F_m and y a nonempty prefix of F_m. Then both x and y are borders of F_m. It holds that $|x| + |y| = |F_{m-2}|$, but as both x and y are nonempty $|x| \leq |F_{m-4}|$, $|y| \leq |F_{m-4}|$ and so there exists no such occurrence of F_m in F_n.

- xy, where x is a nonempty suffix of F_m and y a nonempty prefix of F_m(note that there is no such occurrence in the F_{n-2}, F_{n-1}

expansion). Then both x and y are borders of F_m. It holds that $|x| + |y| = |F_m|$, but as both x and y are nonempty $|x| \leq |F_{m-2}|$, $|y| \leq |F_{m-2}|$ and so there exists no such occurrence of F_m in F_n.

□

Lemma 6.1.8 *For every integer $n \geq 5$, $F_n[1..|F_{n-1}| - 1]$ is not a left seed of F_n.*

Proof Using the recurrence relation we can expand F_n , $n \geq 5$, in the following two ways:

$F_n = F_{n-2}F_{n-3}F_{n-2} = F_{n-2}F_{n-2}F_{n-5}F_{n-4}$

Then one can see that $x = F_n[1..|F_{n-1}| - 1] = F_{n-2}P_{n-3}\delta_{n-3}[1]$ (Lemma 6.1.5). Using Lemma 6.1.7 we can see that by expanding x from the prefix and suffix positions of F_{n-2} we cover F_n except $F_n[|F_{n-1}|]$. Expanding F_{n-2} from its middle occurrence yields the factor $y = F_{n-2}F_{n-5}$ $P_{n-4}\delta_{n-4}[1] = F_{n-2}P_{n-3}\delta_{n-4}[1]$. It is easy to see that x and y differ at their last letter and hence the above result follows.

□

Lemma 6.1.9 *For every integer $n \geq 5$, xF_{n-4}, where x is a suffix of F_{n-3} and $0 < |x| < F_{n-3}$, is not a right seed of F_n.*

Proof Using the recurrence relation we can expand F_n, $n \geq 5$, in the following way:

$F_n = F_{n-4}F_{n-5}F_{n-4}F_{n-4}F_{n-5}F_{n-4}F_{n-5}F_{n-4}$

Then any right seed of the form xF_{n-4}, $0 < |x| < |F_{n-3}|$, has x as a suffix of $F_{n-4}F_{n-5}$. Clearly the 3 occurrences of F_{n-4} at the starting positions of F_{n-5} (Lemma 6.1.7) cannot be expanded to their left to give right seeds as an F_{n-4} is to their left, which has a different ending than that of F_{n-5} (Lemma 6.1.5). Then $F_n[|F_{n-4}| + |F_{n-5}| + |F_{n-4}| + 1]$ cannot be covered by expanding the other 5 occurrences of F_{n-4} in F_n. □

Lemma 6.1.10 *There is no $F_{m-1}F_{m-1}$ in the F_m, F_{m-1} expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-1\}$.*

Proof Any F_{m-1} in the expansion comes from an expanded $F_{m+1} = F_mF_{m-1}$. Therefore, any F_{m-1} in the F_m, F_{m-1} expansion of F_n must be preceded by a F_m. □

Lemma 6.1.11 *There is no $F_mF_mF_m$ in the F_m, F_{m-1} expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-1\}$.*

Proof The above statement holds for $m = n-1$, since $|F_n| < 3|F_{n-1}|$. For $m < n-1$, Lemma 6.1.10 shows that any F_m in the F_{m+1}, F_m expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-2\}$, is preceded by a F_{m+1} and followed by a F_{m+1} or nothing, giving the following cases:

- Expanding $F_{m+1}F_mF_{m+1}$ to $F_mF_{m-1}F_mF_mF_{m-1}$ gives no $F_mF_mF_m$ in the expansion as $F_mF_{m-1}F_mF_mF_{m-1}$ contains no $F_mF_mF_m$ and it can be preceded by at most one F_m (Lemma 6.1.10).
- Expanding $F_{m+1}F_m$ to $F_mF_{m-1}F_m$ gives no $F_mF_mF_m$ in the expansion as $F_mF_{m-1}F_m$ contains no $F_mF_mF_m$ and it can be preceded by at most one F_m (Lemma 6.1.10).

□

The following lemmas show some facts regarding the Fibonacci sequence that will prove useful later on the solution of the problems that we are considering.

Lemma 6.1.12 *[99] The sum of the first n Fibonacci numbers is the $(n+2)^{th}$ Fibonacci number minus 1, i.e. $\sum_{i=1}^{n} f_i = f_{n+2} - 1$.*

Lemma 6.1.13 *[99] The sum of the squares of the first n Fibonacci numbers is the product of the n^{th} and the $(n+1)^{th}$ Fibonacci numbers, i.e. $\sum_{i=1}^{n} f_i^2 = f_n f_{n+1}$.*

Lemma 6.1.14 *[99] The ratio of successive Fibonacci numbers* $\frac{f_n}{f_{n-1}}$ *approaches* $\phi = \frac{1+\sqrt{5}}{2}$*, the golden ratio, as* n *approaches infinity, i.e.* $\lim_{n\to+\infty} \frac{f_n}{f_{n-1}} = \phi$.

Lemma 6.1.15 *[99]* $\sum_{i=1}^{2n} f_i f_{i-1} = f_{2n}^2$.

Lemma 6.1.16 *[99]* $\sum_{i=1}^{2n+1} f_i f_{i-1} = f_{2n+1}^2 - 1$.

6.1.2 Quasiperiodicities in Fibonacci strings

In this section we identify quasiperiodicities in Fibonacci strings (left seeds, right seeds, seeds, covers).

Identifying all covers of a Fibonacci string is made possible by identifying the longest cover of the string and then applying Lemma 6.3.2 as shown in the theorem below.

Theorem 6.1.17

$$\textit{All covers of } F_n \textit{ are:} \begin{cases} F_n, & n \in \{0,1,2,3,4\} \\ \{F_n, F_{n-2}, F_{n-4}, F_{n-6}, \ldots F_3\}, & n = 2k+1, k \geq 2 \\ \{F_n, F_{n-2}, F_{n-4}, F_{n-6}, \ldots F_4\}, & n = 2k, k \geq 3 \end{cases} \tag{6.2}$$

Proof It is easy to see that the theorem holds for $n \in \{0,1,2,3,4\}$. Using the recurrence relation we can expand F_n , $n \geq 5$, in the following two ways:

$F_n = F_{n-2}F_{n-3}F_{n-2} = F_{n-2}F_{n-2}F_{n-5}F_{n-4}$

It is now obvious that F_{n-2} is a cover of F_n. By Lemma 6.1.3 F_{n-2} is also the longest border of F_n and therefore the second longest cover of F_n (after F_n). Similarly F_{n-4} is the longest cover of F_{n-2}, F_{n-6} is the longest cover of F_{n-4}, etc. Hence by following Lemma 6.3.2 we get the above result. □

Corollary 6.1.18

$$\textit{The number of covers of } F_n \textit{ is: } \begin{cases} 1, & n \in \{0,1,2,3,4\} \\ \frac{n-1}{2}, & n = 2k+1, k \geq 2 \\ \frac{n}{2} - 1, & n = 2k, k \geq 3 \end{cases} \tag{6.3}$$

Identifying left seeds of a Fibonacci string F_n is made possible for large n by characterizing each possible left seed as a factor of the form $F_m x$, where $m \in \{3, \ldots n-1\}$ and x a possibly empty prefix of F_{m-1}. We then use the (F_m, F_{m-1})-expansion of F_n along with Lemma 6.1.8 and the following result follows.

Theorem 6.1.19 *All left seeds of F_n are:*

- F_n, *if* $n \in \{0,1,2\}$
- $\{ab, aba\}$, *if* $n = 3$
- $\{F_{n-1}x$: *x a possibly empty prefix of* $F_{n-2}\} \cup_{m=3}^{n-2} \{F_m x$: *$x$ a possibly empty prefix of* $F_{m-1}[1\,..\,|F_{m-1}| - 2]\}$, *if* $n \geq 4$

Proof It is easy to see that the theorem holds for $n \in \{0,1,2,3,4\}$.

For even $n \geq 5$ by Theorem 6.1.17 $\{F_n, F_{n-2}, F_{n-4}, \ldots F_4\}$ are covers of F_n and therefore left seeds of F_n. Again by Theorem 6.1.17 $\{F_{n-1}, F_{n-3}, F_{n-5}, \ldots F_3\}$ are all covers of F_{n-1} which is the period of F_n and hence by Lemma 4.1.1 $\{F_n, F_{n-1}, F_{n-2}, \ldots F_3\}$ are left seeds of F_n. By making similar observations for odd $n \geq 5$ we get that $\{F_n, F_{n-1}, F_{n-2}, \ldots F_3\}$ are all left seeds of F_n in either case. Only a and ab might be shorter left seeds but they are rejected as they are not left seeds of F_4 and so they are not left seeds of any longer Fibonacci string (F_4 is a prefix of every other F_n, $n \geq 5$). Therefore the remaining left seeds are of the form $F_m x$, where $m \in \{3, 4, \ldots, n-1\}$ and $0 < |x| < |F_{m-1}|$. Using the

recurrence relation we can get the (F_m, F_{m-1})-expansion of F_n for any $m \in \{3, 4, \ldots, n-1\}$ as shown before:

$F_n = F_m F_{m-1} F_m F_m F_{m-1} F_m F_{m-1} F_m \ldots$

We then try to expand the seed from each F_m, F_{m-1} in the above expansion as of Lemma 6.1.7 (note that there are no consecutive F_{m-1} in the above expansion).

$F_m F_{m-1} = F_m P_{m-1} \delta_{m-1}$

$F_m F_m = F_m F_{m-1} F_{m-2} = F_m P_{m-1} \delta_{m-1} F_{m-2}$

$F_{m-1} F_m = F_{m-1} F_{m-2} F_{m-3} F_{m-2} = F_m F_{m-3} P_{m-2} \delta_{m-2} = F_m P_{m-1} \delta_{m-2}$

It is now obvious that any $F_m x$, where $m \in \{3, 4, \ldots, n-1\}$ and $0 < |x| \leq |F_{m-1}| - 2$ is a left seed of F_n. $F_{n-1} F_{n-2}[1 \,..\, |F_{n-2}| - 1]$ is the only other left seed as it covers the period of F_n (Lemma 4.1.1). That there are no left seeds of form $F_m x$, where $m \in \{3, 4, \ldots, n-2\}$ and $|x| = |F_{m-1}| - 1$ follows from Lemma 6.1.8.

□

Corollary 6.1.20

$$\textit{The number of left seeds of } F_n \textit{ is: } \begin{cases} 1, & n \in \{0, 1, 2\} \\ 2, & n \in \{3\} \\ |F_n| - n + 3, & n \geq 4 \end{cases} \tag{6.4}$$

Identifying right seeds of a Fibonacci string F_n is made possible for large n by characterizing each possible right seed as a factor of the form xF_m, where $m \in \{3, 5, \ldots n-2\}$ if n is odd or $m \in \{4, 6, \ldots n-2\}$ if n is even, and x is a possibly empty suffix of F_{m+1}. We then use the (F_m, F_{m-1})-expansion of F_n along with Lemma 6.1.9 and the following result follows.

Theorem 6.1.21 *All right seeds of F_n are:*

- F_n, *if* $n \in \{0, 1, 2\}$
- $\{F_n, F_{n-2}, F_{n-4}, F_{n-6}, \ldots F_3\} \cup \{xF_{n-3}F_{n-2}$*: x a possibly empty suffix of* $F_{n-2}\}$, *if* $n = 2k + 1$, $k \geq 1$
- $\{F_n, F_{n-2}, F_{n-4}, F_{n-6}, \ldots F_4\} \cup \{xF_{n-3}F_{n-2}$*: x a possibly empty suffix of* $F_{n-2}\}$, *if* $n = 2k$, $k \geq 2$

Proof It is easy to see that the theorem holds for $n \in \{0, 1, 2, 3, 4\}$. For even $n \geq 5$, by Theorem 6.1.17 $\{F_n, F_{n-2}, F_{n-4}, \ldots F_4\}$ are covers of F_n and therefore right seeds of F_n. Only $\{baab, aab, ab, b\}$ might be shorter right seeds but they are rejected as they are not right seeds of F_6 and so they are not right seeds of any F_n, where n even and $n \geq 5$ (F_6 is a suffix of every other F_n, n even and $n \geq 5$). Similarly for odd $n \geq 5$ $\{F_n, F_{n-2}, F_{n-4}, \ldots F_3\}$ are right seeds of F_n and F_3 is its shortest right seed. Therefore the remaining right seeds are of the form xF_m, where $0 < |x| < |F_{m+1}|$ and $m \in \{4, 6, \ldots, n-2\}$, if n is even, or $m \in \{3, 5, \ldots, n-2\}$, otherwise.

The only other right seeds are of the form $xF_{n-3}F_{n-2}$,where x is a suffix of F_{n-2} and $0 \leq |x| < F_n$, as it is easy to see that they cover the period of F_n (Lemma 4.1.2).

The fact that there are no right seeds of form xF_{n-2}, where $0 < |x| < |F_{n-3}|$, follows from Lemma 6.1.7. Clearly the middle occurrence of F_{n-2} cannot be expanded to the left as an F_{n-2} is to its left, which has a different ending than that of F_{n-3} at the left of the last F_{n-2}. Then $F_n[|F_{n-2}| + 1]$ cannot be covered by expanding the other 2 occurrences of F_{n-2} in F_n.

The fact that there are no right seeds of the form xF_m, where $0 < |x| < |F_{m+1}|$ and $m \in \{4, 6, \ldots, n-4\}$, n is even, or $m \in \{3, 5, \ldots, n-4\}$, otherwise, follows from Lemma 6.1.9. □

Corollary 6.1.22

$$\textit{The number of right seeds of } F_n \textit{ is: } \begin{cases} 1, & n \in \{0,1,2\} \\ |F_{n-2}| + \frac{n-1}{2}, & n = 2k+1, k \geq 1 \\ |F_{n-2}| + \frac{n}{2} - 1, & n = 2k, k \geq 2 \end{cases} \tag{6.5}$$

Identifying all seeds of a Fibonacci string F_n is made possible for large n by characterizing each possible seed as a factor of the form xF_my, where $m \in \{3, 4, \ldots n-1\}$ and x, y follow some restrictions such that F_m is the longest Fibonacci factor in the seed and no occurrence of F_m in the seed starts from a position in x. We then use the (F_m, F_{m-1})-expansion of F_n along with Lemma 6.1.7 and the result below follows.

Theorem 6.1.23 *All seeds of F_n are:*

- *all left/right seeds of F_n, if $n \in \{0,1,2,3\}$*
- *all left/right seeds of F_n and baa, if $n = 4$*
- *all left/right seeds of F_n,*

  strings of form $\{xF_my$: x a suffix of F_m,y a prefix of F_{m-1}, $0 < |x| < |F_m|$, $0 < |y| < |F_{m-1}| - 1$, $|x| + |y| \geq F_{m-1}$ and $m \in \{3, \ldots, n-3\}\}$,

  strings of form $\{xF_{m-1}F_my$: x a suffix of F_m,y a prefix of F_{m-1}, $|x| + |y| \geq F_m$ and $m \in \{3, \ldots, n-3\}\}$,

  strings of form $\{xF_{n-2}y$: x a suffix of F_{n-2},y a prefix of $F_{n-5}F_{n-4}$, $0 < |x| < |F_{n-2}|$, $0 < |y| \leq |F_{n-3}|$ and $|x| + |y| \geq |F_{n-3}|\}$, if $n \geq 5$

Proof It is easy to see that the theorem holds for $n \in \{0,1,2,3,4\}$. For $n \geq 5$ it is obvious that all left seeds of F_n and all right seeds of F_n are also seeds of F_n.

Therefore the remaining seeds are of the form xF_my, such that F_m is the

leftmost occurrence of the longest Fibonacci string present in the seed, $m \in \{3, 4, \ldots, n-2\}$, $|x| > 0$ and $|y| > 0$.

For $m = n-2$ the expansion of $F_n = F_{n-2}F_{n-3}F_{n-2} = F_{n-2}F_{n-2}F_{n-5}F_{n-4}$ is very small so we consider it separately. By expanding the middle occurrence of F_{n-2} we get the seed $xF_{n-2}y$, where $0 < |x| < |F_{n-2}|$, $0 < |y| \leq |F_{n-3}|$ and $|x| + |y| \geq |F_{n-3}|$. As of Lemma 6.1.7 the remaining seeds of form xF_my, such that F_m is the leftmost occurrence of the longest Fibonacci string present in the seed, $m \in \{3, 4, \ldots, n-3\}$, $|x| > 0$ and $|y| > 0$, have their leftmost F_m factor occurring in the start position of either an F_m or an F_{m-1} in the (F_m, F_{m-1})-expansion of $F_n = F_mF_{m-1}F_mF_mF_{m-1}F_mF_{m-1}F_m\ldots$ We consider the following cases (note that there are no consecutive F_{m-1} in the above expansion):

- A seed of form xF_my, such that F_m has a F_{m-1} to its left in the (F_m, F_{m-1})-expansion of F_n and $0 < |x| < F_{m-1}$ (otherwise there exist a new leftmost occurrence of F_m in the seed). The occurrences of F_m that we are considering have starting positions only from a F_m in the expansion of F_n, then y can be up to $F_{m-1}[1\,..\,|F_{m-1}| - 1]$ (otherwise a F_{m+1} is created). But such a seed fails to cover $F_n[|F_mF_{m-1}F_mF_{m-1}| - 1]$.

- A seed of form xF_my, such that F_m has a F_m to its left in the (F_m, F_{m-1})-expansion of F_n and $0 < |x| < F_m$ (otherwise a F_{m+1} is created). If the occurrences of F_m that we are considering have starting positions both from a F_m and a F_{m-1} in the expansion of F_n, then y can be up to $F_{m-1}[1\,..\,|F_{m-1}| - 2]$ (otherwise the factors differ). Furthermore $|x| + |y| \geq |F_{m-1}|$,such as to cover $F_n[|F_mF_{m-1}| + 1\,..\,|F_mF_{m-1}F_m|]$. Such a seed covers F_n as $F_{m+2} = F_mF_{m-1}F_m = F_mF_mP_{m-1}\delta_{m-2}$ is a left seed of F_n (Theorem 6.1.19) composing F_n with concatenations of overlap 0 (factors are joined by considering the seed that its leftmost F_m starts from the next

F_{m+2}) or F_m (factors are joined as $|x| + |y| \geq F_{m-1}$). If the occurrences of F_m that we are considering have starting positions only from a F_m in the expansion of F_n, then y can be up to $F_{m-1}[1\,..\,|F_{m-1}| - 1]$ (otherwise a F_{m+1} is created). But such a seed fails to cover $F_n[|F_mF_{m-1}|]$. If the occurrences of F_m that we are considering have starting positions only from a F_{m-1} in the expansion of F_n, then $|y|$ can be up to $2|F_{m-1}|-1]$ (otherwise a F_{m+1} is created). Furthermore $|x| + |y| \geq |F_m| + |F_{m-1}| = |F_{m+1}|$,such as to cover $F_n[|F_mF_{m-1}|+1\,..\,|F_mF_{m-1}F_mF_m|]$. Such a seed covers F_n as $F_{m+2} = F_mF_{m-1}F_m = F_mF_mP_{m-1}\delta_{m-2}$ is a left seed of F_n (Theorem 6.1.19) composing F_n with concatenations of overlap 0 (factors are joined as $|x| + |y| \geq |F_{m+1}|$) or F_m (factors are joined as $|x| + |y| \geq |F_{m+1}| > |F_{m-1}|$).

□

Corollary 6.1.24 *The number of seeds of F_n is $\Omega(|F_n|^2)$.*

6.1.3 Quasiperiodicities in circular Fibonacci strings

Finding all covers of a circular Fibonacci string is now obvious, we just need to check the seeds of the relevant Fibonacci string. Those which are covers of a superstring of form xF_ny, where x is a possibly empty suffix of F_n and y is a possibly empty prefix of F_n are covers of $C(F_n)$.

Theorem 6.1.25 *All covers of $C(F_n)$ are:*

- *F_n, if $n \in \{0, 1, 2, 3\}$*
- *F_n and F_{n-1}, if $n = 4$*
- *F_n, strings of form $\{F_mx$: x a possibly empty prefix of $F_{m-1}[1\,..\,|F_{m-1}| - 2]$ and $m \in \{3, \ldots, n-1\}\}$,*

strings of form $\{xF_m y$*: x a suffix of* F_m*, y a prefix of* F_{m-1}*,* $0 < |x| < |F_m|$*,* $0 < |y| < |F_{m-1}| - 1$*,* $|x| + |y| \geq F_{m-1}$ *and* $m \in \{3, \ldots, n-2\}\}$*,*

strings of form $\{xF_{m-1}F_m y$*: x a suffix of* F_m*, y a prefix of* F_{m-1}*,* $|x| + |y| \geq F_m$ *and* $m \in \{3, \ldots, n-3\}\}$*, if* $n \geq 5$

Proof It is easy to see that the theorem holds for $n \in \{0, 1, 2, 3, 4\}$. For larger n the covers of $C(F_n)$ are at most the seeds of F_n. A seed is a cover of $C(F_n)$ iff it covers a superstring of F_n of form $xF_n y$, where x is a possibly empty suffix of F_n and y is a possibly empty prefix of F_n. We consider the following cases:

- Left seeds of form $F_n[1\,..\,|F_k| + i]$, where $i \in \{0, 1, \ldots, |F_{k-1}| - 2\}$ and $k \in \{3, 4, \ldots n-1\}$, are covers of $F_n F_k[1\,..\,i]$,if F_k is a cover of F_n, or covers of $F_n F_k[1\,..\,|F_{k-2}| + i]$ otherwise, and hence covers of $C(F_n)$ in both cases. Clearly F_n is also a cover of $C(F_n)$. $F_n[1\,..\,|F_n| - 1]$ is not a cover of $C(F_n)$ as it fails to cover a prefix of $F_n F_n$ longer than $|F_n| - 1$ (consider the F_{n-1}, F_{n-2} expansion of F_n along with Lemma 6.1.7).

- The only right seeds of F_n that are covers of $C(F_n)$ are the covers of F_n (included above). Right seeds of form $xF_{n-3}Fn-2$, where x is a possibly empty suffix of F_{n-2} and $0 \leq |x| < |F_{n-2}|$, fail to cover a suffix of $F_n F_n = Fn-2F_{n-3}Fn-2$ longer than $|xF_{n-3}F_{n-2}|$ (consider the F_{n-2}, F_{n-3} expansion of F_n along with Lemma 6.1.7), and so they are not covers of $C(F_n)$.

- Seeds of form $xF_m y$ where x a suffix of F_m and y a prefix of F_{m-1}, $0 < |x| < |F_m|$, $0 < |y| < |F_{m-1}| - 1$, $|x| + |y| \geq F_m$ and $m \in \{3, 4, \ldots, n-3\}$ are covers of $xF_n F_m y$,if F_m is a cover of F_n, or covers of $xF_{m-1}F_n F_{m-2}y$ otherwise, and hence covers of $C(F_n)$ in both cases.

- Seeds of form $xF_{m-1}F_my$ where x a suffix of F_m and y a prefix of F_{m-1}, $0 < |x| < |F_m|$, $0 < |y| < |F_{m-1}|$, $|x| + |y| \geq F_m$ and $m \in \{3, 4, \ldots, n-3\}$ are covers of $xF_{m-1}F_mF_ny$,if F_m is a cover of F_n, or covers of $xF_{m-1}F_nF_my$ otherwise, and hence covers of $C(F_n)$ in both cases.

- Seeds of form $xF_{n-2}y$ where x a suffix of F_{n-2} and y a prefix of $F_{n-5}F_{n-4}$ such that $0 < |x| < |F_{n-2}|$, $0 < |y| < |F_{n-3}| - 1$ and $|x| + |y| \geq |F_{n-3}|$ are covers of $xF_nF_{n-2}y$ and hence covers of $C(F_n)$. When $y = F_{n-5}F_{n-4}$ or $F_{n-5}F_{n-4}[1\,..\,|F_{n-4}| - 1]$ the seed fails to cover $xF_nF_{n-2}y$, the first F_{n-2} of F_n cannot be expanded further to the right. Trying to force an overlap of $xF_{n-2}y$ to the left of $F_nF_{n-2}y$ gives the superstrings $xF_{n-3}F_nF_{n-2}y$ and $xF_{n-2}F_{n-5}F_nF_{n-2}y$ (consider the occurrences of F_{n-4} in F_n), which are not made of suffixes of F_n, as clearly F_{n-3} and $Fn-5$ are not borders of F_n (Lemma 6.1.3), and so they are not covers of $C(F_n)$.

□

Corollary 6.1.26 *The number of covers of $C(F_n)$ is $\Omega(|C(F_n)|^2)$.*

6.1.4 Bounds on the number of seeds of a string

It is easy to see that the number of seeds a string x is bounded above by $\frac{|x|^2}{2}$.

Theorem 6.1.27 *The number of distinct seeds of a nonempty string x is at most $\frac{n(n+1)}{2}$, where $|x| = n$.*

Proof The number of non-empty factors of x is $n + \binom{n}{2} = \frac{n(n+1)}{2}$. □

We have seen that the number of seeds a Fibonacci string F_n is $\Omega(|F_n|^2)$ (Theorem 6.1.23). The following theorem proves that as $n \to$

$+\infty$ the ratio of the number of distinct seeds of F_n to the square of its length converges to $\frac{\phi^2+1}{2\phi^6} = 0.100813061875578\ldots$, i.e. still not very close to $\frac{1}{2}$.

Theorem 6.1.28 $\lim_{n\to+\infty} \frac{Seeds(F_n)}{|F_n|^2} = \frac{\phi^2+1}{2\phi^6}$

Proof Summing all the seeds of Theorem 6.1.23 and considering only the quadratic terms we get:

$$\lim_{n\to+\infty} \frac{\text{Seeds}(F_n)}{|F_n|^2}$$
$$= \lim_{n\to+\infty} \sum_{m=0}^{n-3} \frac{|F_m|^2 + 2|F_m||F_{m-1}|}{2|F_n|^2}$$
$$= \lim_{n\to+\infty} \sum_{m=0}^{n-3} \frac{2|F_m|^2 + |F_{m+1}|^2 - |F_{m-2}|^2}{4|F_n|^2}$$
$$= \lim_{n\to+\infty} \frac{2|F_{n-3}||F_{n-2}| + |F_{n-2}||F_{n-1}| - |F_{n-5}||F_{n-4}|}{4|F_n|^2}$$
(as $\sum_{i=0}^{n} |F_i|^2 = |F_n||F_{n+1}|$)
$$= \lim_{n\to+\infty} \frac{1}{4|F_n|^2}(2|F_{n-3}||F_{n-2}| + |F_{n-2}|(|F_{n-2}| + |F_{n-3}|)$$
$$-(2|F_{n-3}| - |F_{n-2}|)(|F_{n-2}| - |F_{n-3}|))$$
$$= \lim_{n\to+\infty} \frac{|F_{n-2}|^2 + |F_{n-3}|^2}{2|F_n|^2}$$
$$= \tfrac{\phi^{-4}+\phi^{-6}}{2} \text{ (as } \lim_{n\to+\infty} \frac{F_n}{F_{n-1}} = \phi)$$
$$= \tfrac{\phi^2+1}{2\phi^6}$$
$$= 0.100813061875578\ldots$$

□

6.1.5 Overlapping factors in Fibonacci words

In this section we investigate the appearance of overlapping factors in Fibonacci words. By considering the expansion of a Fibonacci string as a concatenation of two consecutive Fibonacci substrings we are able to identify Fibonacci subwords in a Fibonacci word and the positions of their occurrences, thus being able to derive information for the overlapping factors of Fibonacci words. More specifically we identify all overlapping factors of a Fibonacci word and we provide a limit for the ratio of the

number of distinct overlapping factors in a Fibonacci word F_n over the square of its length.

The following lemmas identify some overlapping factors in Fibonacci words by looking in the expansion of Fibonacci strings as a concatenation of two consecutive Fibonacci substrings.

This lemma shows that factors of F_n of form xF_my, where x is a non empty suffix of F_m, y is a non-empty prefix of F_{m-1}, $1 \leq |x| \leq |F_m|-1$and $1 \leq |y| \leq |F_{m-1}| - 2$ are overlapping factors of F_n in certain cases.

Lemma 6.1.29 *Factors of F_n of form xF_my, where x is a non empty suffix of F_m, y is a non-empty prefix of F_{m-1}, $1 \leq |x| \leq |F_m| - 1$, $1 \leq |y| \leq |F_{m-1}| - 2$ and $3 \leq m \leq n-4$, are overlapping factors of F_n for $n \geq 6$.*

Proof We consider the F_m, F_{m-1} expansion of F_n. Suppose that there are two occurrences of the required factor (see also Figure 6.1) that overlap. There are three cases:

- The F_m that appears in the second factor starts k positions ($1 \leq k \leq |F_m| - 1$) after the F_m that appears in the first factor. As in Lemma 6.1.7 k can only be $|F_{m-1}|$. Then the factors $x_1F_my_1$ and $x_2F_my_2$ form a quasiperiodic square of the form $x_1F_{m-1}Fmy_2$, which is a contradiction as x_2 is a suffix of F_{m-1}.

- The F_m that appears in the first factor and the F_m that appears in the second factor are consecutive. Hence overlapping factors of this form must lie in a quasiperiodic square of the form $F_mF_mF_{m-1}F_m$. Then y can be up to:

$$\begin{aligned} LCP(F_{m-1}, F_{m-3}F_{m-2}) &= LCP(P_{m-1}\delta_{m-1}, F_{m-3}P_{m-2}\delta_{m-2}) \\ &= LCP(P_{m-1}\delta_{m-1}, P_{m-1}\delta_{m-2}) \\ &= P_{m-1} \end{aligned}$$

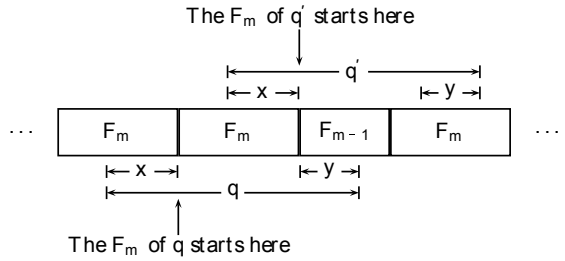


Figure 6.1: Factors of F_n, q and q' (with $q = q'$), of the form xF_my, where x is a non empty suffix of F_m and y is a non empty prefix of F_{m-1}. Notice that the occurrences of F_m in q and q' are consecutive.

Clearly, $1 \leq |x| \leq |F_m| - 1$ and $1 \leq |y| \leq |F_{m-1}| - 2$.

- The F_m that appears in the first factor and the F_m that appears in the second factor are separated by a non-empty factor z. Hence overlapping factors of this form must lie in a factor u of the form xF_mzF_my, where z is a non empty word. Then $z = vF_m$ which forces y to be of the form F_mw or $F_{m-1}s$ which contradicts the length of y (v, w, s are non empty words).

□

The following lemma shows that factors of F_n of the form xF_my, where x is a non empty suffix of F_{m-1} and y is a non empty prefix of F_{m-1}, with $1 \leq |x| \leq |F_{m-1}| - 1$, $1 \leq |y| \leq |F_{m-1}| - 1$ and $|x| + |y| > |F_{m-1}|$ are overlapping factors of F_n in certain cases.

Lemma 6.1.30 *Factors of F_n of the form xF_my, where x is a non empty suffix of F_{m-1} and y is a non empty prefix of F_{m-1}, with $1 \leq |x| \leq |F_{m-1}| - 1$, $1 \leq |y| \leq |F_{m-1}| - 1$, $|x| + |y| > |F_{m-1}|$ and $3 \leq m \leq n - 5$, are overlapping factors of F_n for $n \geq 8$.*

Proof Lemmas 6.1.7, 6.1.10 and 6.1.11 suggest that quasiperiodic squares containing overlapping factors of the required form must lie in a factor of the form $F_{m-1}F_mF_{m-1}F_mF_{m-1}$ in the F_mF_{m-1} expansion of F_n. Clearly,

$1 \leq |x| \leq |F_m| - 1$ and $1 \leq |y| \leq |F_{m-1}| - 1$ with the restriction that $|x| + |y| > |F_{m-1}|$, so that the factors overlap. □

The following lemma shows that factors of F_n of the form xF_my, where x is a non empty suffix of F_{m-1}, y is a non empty prefix of F_m and $2 \leq m \leq n-4$, are generally not overlapping factors of F_n.

Lemma 6.1.31 *Factors of F_n of the form xF_my, where x is a non empty suffix of F_{m-1}, y is a non empty prefix of F_m and $2 \leq m \leq n-4$, are not overlapping factors of F_n for $n \geq 6$.*

Proof Lemmas 6.1.7, 6.1.10 and 6.1.11 suggest that quasiperiodic squares containing overlapping factors of the required form must lie in a factor of the form $F_{m-1}F_mF_mF_{m-1}F_mF_m$ in the F_mF_{m-1} expansion of F_n but this causes no overlap of the squares. □

The following lemma shows that factors of F_n of the form $xF_{m-1}y$, where x is a non empty suffix of F_m and y is a non empty prefix of F_{m+1}, with $1 \leq |x| \leq |F_m| - 1$, $|F_{m-2}| + 1 \leq |y| \leq |F_{m+1}| - 1$, $|x| + |y| > |F_{m-1}|$ and $3 \leq m \leq n-5$, are overlapping factors of F_n in certain cases.

Lemma 6.1.32 *Factors of F_n of the form $xF_{m-1}y$, where x is a non empty suffix of F_m and y is a non empty prefix of F_{m+1}, with $1 \leq |x| \leq |F_m| - 1$, $|F_{m-2}| + 1 \leq |y| \leq |F_{m+1}| - 1$, $|x| + |y| > |F_{m-1}|$ and $3 \leq m \leq n-5$, are overlapping factors of F_n for $n \geq 8$.*
Factors of F_n of the form $xF_{n-5}y$, where x is a non empty suffix of F_{n-4} and y is a non empty prefix of F_{n-3}, such that either ($1 \leq |x| \leq |F_{n-4}| - 1$, $|F_{n-6}| + 1 \leq |y| \leq |F_{n-4}| - 2$ and $|x| + |y| > |F_{n-5}|$) or ($1 \leq |x| \leq |F_{n-4}| - 1$, $|F_{n-6}| + 1 \leq |y| \leq |F_{n-3}| - 1$, $|x| + |y| > |F_{n-3}|$), are overlapping factors of F_n for $n \geq 6$.

Proof Lemmas 6.1.7, 6.1.10 and 6.1.11 suggest that quasiperiodic squares containing overlapping factors of the required form must lie in a factor of the F_mF_{m-1} expansion of F_n having one of the following forms:

- $F_mF_{m-1}F_mF_{m-1}F_mF_m$, where $3 \leq m \leq n-5$.
  Clearly, $1 \leq |x| \leq |F_m| - 1$ and $|F_{m-2}| + 1 \leq |y| \leq |F_{m+1}| - 1$ with the restriction that $|x| + |y| > |F_{m-1}|$.

- $F_mF_{m-1}F_mF_mF_{m-1}$, where $3 \leq m \leq n-4$.
  Clearly, $1 \leq |x| \leq |F_m| - 1$ and $|F_{m-2}| + 1 \leq |y| \leq |F_m| - 2$ with the restriction that $|x| + |y| > |F_{m-1}|$.

- $F_mF_{m-1}F_mF_mF_{m-1}F_mF_{m-1}$, where $3 \leq m \leq n-4$.
  Clearly, $1 \leq |x| \leq |F_m| - 1$ and $|F_{m-2}| + 1 \leq |y| \leq |F_{m+1}| - 1$ with the restriction that $|x| + |y| > |F_{m+1}|$.

□

The following lemma gives all the overlapping factors in a Fibonacci word by identifying factors in the F_m, F_{m-1} expansion of F_n, where $n \in \{2, 3, \dots\}$ and $m \in \{1, 2, \dots, n-1\}$, and by using the results of Lemmas 6.1.29, 6.1.30, 6.1.31 and 6.1.32.

Lemma 6.1.33 *The overlapping factors of F_n are:*

- $\emptyset$, *if* $n = \{0, 1, 2, 3\}$

- *factors of the form F_my, where y is a possibly empty prefix of F_{m-1}, $0 \leq |y| \leq |F_{m-1}| - 1$ and $3 \leq m \leq n-4$, factors of F_n of the form xF_m, where x is a possibly empty prefix of F_m, $0 \leq |x| \leq |F_m| - 1$ and $3 \leq m \leq n-4$ and factors mentioned in Lemmas 6.1.29, 6.1.30 and 6.1.32, if $n \geq 6$.*

Proof It is easy to see that the lemma is valid for $0 \leq n \leq 5$.

For larger n, we observe that any overlapping factors are of the form xF_my, such that F_m is the leftmost occurrence of the longest Fibonacci word present in the factor, with $m \in \{3, 4, \ldots, n-1\}$, $|x| \geq 0$ and $|y| \geq 0$. Clearly, for $m = n-1$ there are no factors of the above form that compose a quasiperiodic square. (Lemma 6.1.7 shows that F_{n-1} occurs in F_n only as a prefix of it).

For $m = n-2$, consider the F_{n-2}, F_{n-3} expansion of F_n: $F_n = F_{n-2}F_{n-3}F_{n-2} = F_{n-2}F_{n-2}F_{n-5}F_{n-4}$. Then $x = 0$ and y can be up to

$$\begin{aligned} LCP(F_{n-2}, F_{n-5}F_{n-4}) &= LCP(P_{n-3}\delta_{n-3}, F_{n-5}P_{m-4}\delta_{n-4}) \\ &= LCP(P_{n-3}\delta_{n-3}, P_{n-3}\delta_{n-4}) \\ &= P_{n-3} \end{aligned}$$

For $m = n-3$, consider the F_{n-3}, F_{n-4} expansion of F_n: $F_n = F_{n-3}F_{n-4}F_{n-3}F_{n-3}F_{n-4} = F_{n-3}F_{n-3}F_{n-6}F_{n-5}F_{n-3}F_{n-4}$. As before, $x = 0$ and y can be up to $LCP(F_{n-4}, F_{n-6}F_{n-5}) = P_{n-4}$. Considering the second occurrence of F_{n-3} we get quasiperiodic squares composed by factors of the form $xF_{n-3}y$, where x is a possibly empty suffix of F_{n-3} and y is a possibly empty prefix of F_{n-4}, with $0 \leq |x| \leq |F_{n-3}| - 1$, $0 \leq |y| \leq |F_{n-4}| - 1$ and $|x| + |y| > |F_{n-4}|$.

For $m \leq n-4$, we get the following cases:

- Factors of F_n of the form F_my, where y is a possibly empty prefix of F_{m-1}, $0 \leq |y| \leq |F_{m-1}| - 1$ and $3 \leq m \leq n-4$ form quasiperiodic squares of F_n.

- Factors of F_n of the form xF_m, where x is a possibly empty prefix of F_m, $0 \leq |x| \leq |F_m| - 1$ and $3 \leq m \leq n-4$, form quasiperiodic squares of F_n.

- Factors of the form xF_my, where $|x| > 0$ and $|y| > 0$ are given by

Lemmas 6.1.29, 6.1.30, 6.1.31 and 6.1.32. □

The following theorem counts the number of distinct overlapping factors in a Fibonacci word F_n, denoted by $OF(F_n)$ and shows that its limit over the square of its length tends to a constant number.

Theorem 6.1.34 $\lim_{n\to+\infty} \frac{OF(F_n)}{|F_n|^2} = 0.0597933994\ldots$

Proof By summing up the number of overlapping factors given by Lemma 6.1.33 and considering only the quadratic terms (as Lemma 6.1.12 shows that the sum of linear terms gives a linear term) we get:

$$\lim_{n\to+\infty} \frac{\mathrm{OF}(F_n)}{|F_n|^2} = \lim_{n\to+\infty} \frac{1}{|F_n|^2}(f_{n-4}f_{n-5} + \frac{1}{2}f_{n-4}^2 + 2\sum_{i=0}^{n-5} f_i f_{i-1} + f_{n-4}f_{n-5} + \frac{1}{2}f_{n-5}^2 + \frac{1}{2}\sum_{i=0}^{n-6} f_i^2)$$

where the first two terms come from the case of $m = n - 3$ in Lemma 6.1.33, the next three terms come from Lemmas 6.1.29 and 6.1.32 and the final term comes from Lemma 6.1.30. Therefore:

$$\begin{aligned}
\lim_{n\to+\infty} \frac{\mathrm{OF}(F_n)}{|F_n|^2} &= \lim_{n\to+\infty} \frac{1}{f_n^2}(2\sum_{i=0}^{n-4} f_i f_{i-1} + \frac{1}{2}\sum_{i=0}^{n-4} f_i^2) \\
&= \lim_{n\to+\infty} \frac{1}{f_n^2}(2f_{n-4}^2 + \frac{1}{2}f_{n-4}f_{n-3}) \\
&\quad \text{(By applying Lemmas 6.1.13, 6.1.15 and 6.1.16)} \\
&= \frac{2}{\phi^8} + \frac{1}{2\phi^7} \text{ (By applying Lemma 6.1.14)} \\
&= 0.0597933994\ldots
\end{aligned}$$

□

6.2 Abelian borders in Fibonacci words

In this section we investigate the appearance of abelian borders in Fibonacci words. We give the necessary and sufficient condition for a prefix of a Fibonacci word to be an abelian border of it and we reveal how many abelian borders exist in a Fibonacci word by using basic recurrence relations.

Theorem 6.2.1 *$F_n[1\,..\,i]$ is an abelian border of F_n iff $\lfloor\frac{i+1}{\phi}\rfloor = \lfloor\frac{f_n+1}{\phi}\rfloor - \lfloor\frac{f_n-i+1}{\phi}\rfloor$.*

Proof Let $a(x)$ be a function giving the number of letters a that exist in a word x.
As in Lemma 6.1.1:

$$\begin{aligned} a(F_\infty[1\,..\,i]) &= \max\{k : k \in \mathbb{Z}, \lfloor k\phi \rfloor \leq i\} \\ &= \max\{k : k \in \mathbb{Z}, k\phi < i+1\} \\ &= \max\{k : k \in \mathbb{Z}, k < \frac{i+1}{\phi}\} \\ &= \lfloor\frac{i+1}{\phi}\rfloor \end{aligned}$$

$F_n[1\,..\,i]$ is an abelian border of F_n iff:

$a(F_n[1\,..\,i]) = a(F_n[f_n{-}i{+}1\,..\,f_n])$, i.e. iff $a(F_\infty[1\,..\,i]) = a(F_\infty[1\,..\,f_n])$ $-a(F_\infty[1\,..\,f_n - i])$. Substituting the previous established relation gives the above result. □

Theorem 6.2.2 *The number of abelian borders of F_n is $-(-1)^n + (1 - \frac{1}{\sqrt{5}})\phi^n + (1 + \frac{1}{\sqrt{5}})(1-\phi)^n - 1$.*

Proof Let $AB(x)$ be a function giving the number of abelian borders of a word x (here we also count x in $AB(x)$ to simplify some calculations

in the next lines). Using the expansion of F_n we get:

$$F_n = F_{n-2}F_{n-3}F_{n-2}$$

Therefore:

$$AB(F_n) = 2AB(F_{n-2}) + AB(F_{n-3})$$

The above relation is a linear recurrence relation which has a solution of the form cr^n. Substituting in the above relation we get:

$$\begin{aligned} r^n = 2r^{n-2} + r^{n-3} &\iff r^{n-3}(r^3 - 2r - 1) = 0 \\ &\iff r^{n-3}(r(r^2-1) - (r+1)) = 0 \\ &\iff r^{n-3}(r+1)(r^2 - r - 1) = 0 \end{aligned}$$

which yields the solutions $r = \{0, -1, \phi, 1-\phi\}$. Hence $AB(F_n) = c_1(-1)^n + c_2\phi^n + c_3(1-\phi)^n$. Substituting the initial conditions ($AB(F_0) = 1$, $AB(F_1) = 1$, $AB(F_2) = 1$) we get the following system of equations:

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & \phi & 1-\phi \\ 1 & \phi^2 & (1-\phi)^2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

By multiplying both sides with the inverse of the first matrix we get: $c_1 = -1$, $c_2 = 1 - \frac{1}{\sqrt{5}}$ and $c_3 = 1 + \frac{1}{\sqrt{5}}$ and hence the above result. (We show the first ten Fibonacci words with their corresponding number of abelian borders in Table 6.1) □

Fibonacci word	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
Number of abelian borders	0	0	0	2	2	6	8	16	24	42

Table 6.1: The first ten Fibonacci words with their corresponding number of abelian borders

6.3 Padovan words

The *nth Padovan number* denoted by p_n is defined as:

$p_0 = 1, \quad p_1 = 1, \quad p_2 = 1, p_n = p_{n-2} + p_{n-3} \quad n \in \{3, 4, 5, \dots\}$

The first few terms are: $1, 1, 1, 2, 2, 3, 4, 5, 7, 9, \dots$

We define a (finite) *Padovan string* P_k , $k \in \{0, 1, 2, \dots\}$, as follows:

$P_0 = c, \;\; P_1 = b, \;\; P_2 = a, \;\; P_n = P_{n-2}P_{n-3}, \;\; n \in \{3, 4, 5, \dots\}$

We mention that $|P_n| = p_n$, the nth Padovan number.

The above sequence can be also defined using the following morphism:

$P_0 = c, \;\; P_n = f(P_{n-1}) = f(P_{n-1}[1])f(P_{n-1}[2]) \dots f(P_{n-1}[|P_{n-1}|]) \;\; n \geq 1$

,where $f(a) = bc, \;\; f(b) = a, \;\; f(c) = b.$

P_0	c
P_1	b
P_2	a
P_3	bc
P_4	ab
P_5	bca
P_6	abbc
P_7	bcaab
P_8	abbcbca
P_9	bcaababbc
P_{10}	abbcbcabcaab

Figure 6.2: The first eleven Padovan words

6.3.1 Properties

In this section, we prove and also quote some properties for the borders and covers of a given string as well as some facts on Padovan strings that will prove useful later on the solution of the problems that we are considering.

The following two lemmas give the relations between borders/covers of a word and provide a way of identifying all borders/covers of a word. Lemma 6.3.1, which we give without proof, is used in almost every publication regarding periodicity.

Lemma 6.3.1 *Let u be a border of x and let $z \neq u$ be a substring of x such that $|z| \leq |u|$. Then z is a border of x if and only if z is a border of u.*

Lemma 6.3.2 *[196] Let u be a cover of x and let $z \neq u$ be a substring of x such that $|z| \leq |u|$. Then z is a cover of x if and only if z is a cover of u.*

Proof Clearly if z is a cover of u and u is a cover of x the z is a cover of x. Suppose now that both z and u are covers of x. Then z is a border of x and hence of u ($|z| \leq |u|$); thus z must also be a cover of u.

Before analysing further the Padovan words we give a proof of equivalence of its different definitions.

Lemma 6.3.3 *The following sequence of strings P'_k , $k \in \{0, 1, 2, \dots\}$, which is defined using the following morphism:*
$P'_0 = c, \quad P'_n = f(P'_{n-1}) = f(P'_{n-1}[1])f(P'_{n-1}[2]) \dots f(P'_{n-1}[|P'_{n-1}|]) \quad n \geq 1$, *where $f(a) = bc, \quad f(b) = a, \quad f(c) = b$, gives us the Padovan sequence of strings.*

Proof We use induction on n. The two definitions agree for $n = \{0, 1, 2\}$. Suppose that they also agree for $n \leq k$, where $k \geq 2$. Then using the first definition we get that $P_{k+1} = P_{k-1}P_{k-2}$. Using the second definition we get that $P'_{k+1} = f(P'_k) = f(P'_{k-2}P'_{k-3}) = f(P'_{k-2})f(P'_{k-3}) = P'_{k-1}P'_{k-2} = P_{k-1}P_{k-2}$ and therefore the two definitions are equivalent for $n \in \{0, 1, 2, \dots\}$.

6.3.2 Borders of Padovan words

In this section we identify all borders of a Padovan word by finding the longest border (Lemma 6.3.4) and then applying Lemma 6.3.1.

Lemma 6.3.4 *The longest border of P_n is P_{n-6}, where $n \geq 7$.*

Proof Use induction on n.
It is easy to see that the above statement is valid for $n \in \{7, 8, 9\}$.
The above statement together with Lemma 6.3.1 suggests that the borders of P_n are:

- no borders, if $n = \{0, 1, 2, 3, 4, 5, 6\}$
- $P_{n-6}, P_{n-12}, \ldots P_{n \bmod 6}$, if $n \geq 7$ and $n \bmod 6 \neq 0$
- $P_{n-6}, P_{n-12}, \ldots P_6$, if $n \geq 7$ and $n \bmod 6 = 0$

For $n > 9$ we consider the following expansion of P_n:
$P_n = P_{n-6}P_{n-7}P_{n-5}P_{n-5}P_{n-6}$
Obviously P_{n-6} is a border of P_n. Suppose there exists a border longer than $|P_{n-6}|$, i.e. a substring of form $P_{n-6}x = yP_{n-6}$, where $|x| = |y|$. Then we get the following cases on x:

- $0 < |x| \leq |P_{n-7}|$: Then we have y a suffix of P_{n-5} and P_{n-2} and a prefix of P_n, P_{n-2}, P_{n-4} and P_{n-6}, therefore a border of P_{n-2}. x is a prefix of P_{n-7}, P_{n-5}, P_{n-3}, P_{n-1} and a suffix of P_n, P_{n-3} and P_{n-6}, therefore a border of P_{n-3}. In order to satisfy the length restrictions $(x, y) \in \{(P_3, P_4), (P_1, P_2), (P_0, P_1)\}$. However in any of those cases x contains more letters ℓ, some $\ell \in \{a, b, c\}$, than y .
- $|P_{n-7}| < |x| \leq |P_{n-5}|$: Then we have y a suffix of P_{n-5} and P_{n-2} and a prefix of P_n, P_{n-2} and P_{n-4}, therefore a border of P_{n-2}. By the inductive hypothesis $|y| \leq |P_{n-8}|$ and therefore we get a contradiction.

- $|P_{n-5}| < |x| \leq |P_{n-5}| + |P_{n-7}|$: Then we have $y = wP_{n-5}$, where w is a suffix of P_{n-5} and P_{n-2} and a prefix of P_n, P_{n-2}, P_{n-4} and P_{n-6}, therefore a border of P_{n-2}. $x = P_{n-7}z$, where z is a prefix of P_{n-5}, P_{n-3}, P_{n-1} and a suffix of P_n, P_{n-3} and P_{n-6}, therefore a border of P_{n-3}. However this contradicts $|w| + |P_{n-5}| = |z| + |P_{n-7}|$ as then $|z| - |w| = P_{n-8}$ and by the inductive hypothesis $|z| \leq P_{n-9}$.

- $|P_{n-5}| + |P_{n-7}| < |x| \leq 2|P_{n-5}|$: Then we have $y = wP_{n-5}$, where w is a suffix of P_{n-5} and P_{n-2} and a prefix of P_n, P_{n-2} and P_{n-4}, therefore a border of P_{n-2}. $x = P_{n-7}P_{n-5}z$, where z is a prefix of P_{n-7}, P_{n-5}, P_{n-3}, P_{n-1} and a suffix of P_n, P_{n-3} and P_{n-6}, therefore a border of P_{n-3}. However this contradicts $|w| + |P_{n-5}| = |z| + |P_{n-5}| + |P_{n-7}|$ as then $|w| - |z| = P_{n-7}$ and by the inductive hypothesis $|w| \leq P_{n-8}$.

- $2|P_{n-5}| < |x| \leq 2|P_{n-5}| + |P_{n-7}|$: Then we have $y = wP_{n-5}P_{n-5}$, where w is a suffix of P_{n-7}, P_{n-4} and P_{n-1} and a prefix of P_n, P_{n-2}, P_{n-4} and P_{n-6}, therefore a border of P_{n-4}. $x = P_{n-7}P_{n-5}z$, where z is a prefix of P_{n-7}, P_{n-5}, P_{n-3}, P_{n-1} and a suffix of P_n, P_{n-3} and P_{n-6}, therefore a border of P_{n-3}. However this contradicts $|w| + 2|P_{n-5}| = |z| + |P_{n-5}| + |P_{n-7}|$ as then $|z| - |w| = P_{n-8}$ and by the inductive hypothesis $|z| \leq P_{n-9}$.

- $2|P_{n-5}|+|P_{n-7}| < |x| < |P_n|-|P_{n-6}|$: Then we have $y = wP_{n-7}P_{n-5}$ P_{n-5}, where w is a suffix of P_{n-6}, P_{n-3} and P_n and a prefix of P_n, P_{n-2}, P_{n-4} and P_{n-6}, therefore a border of P_{n-6}. Similarly x is a prefix of $P_{n-7}P_{n-5}P_{n-5}z$, where z is a prefix of P_{n-6}, P_{n-4}, P_{n-2} and P_n and a suffix of P_n, P_{n-3} and P_{n-6}, therefore a border of P_{n-6}. Considering the first occurrence of P_{n-6} in y we get $P_{n-6} = wP_{n-7}u$, where u is a border of P_{n-9}. Considering the last occurrence of P_{n-6} in x we get $P_{n-6} = vP_{n-8}z$, where v is a border of P_{n-4}. Then

$|w| + |P_{n-7}| + |u| = |v| + |P_{n-8}| + |z|$ and hence $|v| - |u| = |P_{n-12}|$, which means that $v = P_{n-10}$ and $|u| = |P_{n-13}| = |P_{n-15}|$ (inductive hypothesis). The equation $|w| + |P_{n-7}| + |u| = |P_{n-6}|$ together with the above relation gives us $|w| = |P_{n-14}|$. It is now possible to see that there is no such case as $|P_{n-13}| = |P_{n-15}|$ and $|P_{n-12}| = |P_{n-14}|$ (as it implies that both $P_{n-15} = P_0$ and $P_{n-14} = P_0$).

□

Theorem 6.3.5

$$\textit{The borders of } P_n \textit{ are: } \begin{cases} \varnothing, & n \in \{0,1,2,3,4,5,6\} \\ P_{n-6}, P_{n-12} \dots P_{n \bmod 6}, & n \geq 7,\ n \bmod 6 \neq 0 \\ P_{n-6}, P_{n-12}, \dots P_6, & n \geq 7,\ n \bmod 6 = 0 \end{cases} \tag{6.6}$$

Proof Direct consequence of Lemma 6.3.4 and Lemma 6.3.1. □

Corollary 6.3.6

$$\textit{The number of borders of } P_n \textit{ is: } \begin{cases} 0, & n = 0 \\ \lceil \frac{n}{6} \rceil - 1, & n \geq 1 \end{cases} \tag{6.7}$$

6.3.3 Factor structure of Padovan words

In this section we give an elementary analysis of the factors of Padovan words. By making extensive use of Theorem 6.3.5 and by considering the expansion of a Padovan string as a concatenation of three consecutive Padovan substrings we are able to identify Padovan subwords in a Padovan word and the positions of their occurrences, thus getting information for all the factors of the Padovan word.

We begin by investigating the appearance of a, b, c in Padovan words. In fact the number of as/bs/cs in P_n are the Padovan numbers P_{n-4}, P_{n-3} and P_{n-5} respectively. We will only give the proof for as. A similar result is also stated in [237] without proof.

Lemma 6.3.7 *The number of occurrences of the letter $a/b/c$ in P_n is the Padovan number $p_{n-4}/p_{n-3}/p_{n-5}$ respectively, where $n \geq 5$.*

Proof It is easy to verify the above statement for $n \in \{4, 5\}$, where the number of as in $P_4/P_5/P_6$ is $1/1/1$, the first three Padovan numbers. For $n \geq 7$ it holds that $P_n = P_{n-2}P_{n-3}$ and therefore the number of as in P_n is the number of as in P_{n-2} plus the number of as in P_{n-3} (the Padovan recurrence relation). A similar argument gives equivalent results for the number of bs and the number of cs. □

Another remarkable fact is that there are two infinite Padovan words:

$$abbcbcabcaabbcaababbcbcaababbcabbcbcabcaababbcabbcbca\ldots$$

and

$$bcaababbcabbcbcaabbcbcabcaababbcbcabcaabbcaababbcabbc\ldots$$

We remind the reader that the infinite Fibonacci word is an infinite word that has every Fibonacci word as a prefix. Here the first infinite word shown has every even Padovan word as a prefix (except P_0) whereas the second one has every odd Padovan word as a prefix. Both words are created by successive application of the morphism f on the letter c, where $f(a) = bc, \quad f(b) = a, \quad f(c) = b$.

Furthermore it is sometimes useful to consider the expansion of a Padovan string as a concatenation of three Padovan substrings. We define

the P_m, P_{m-1}, P_{m-2} expansion of P_n, where $n \geq 4$ and $m \in \{2, 3 \ldots, n-2\}$, as follows:

- Expand P_n using the recurrence formula resulting to $P_{n-2}P_{n-3}$.
- Expand P_{n-2} using the recurrence formula resulting to $P_{n-4}P_{n-5}$ P_{n-3}.
- Expand P_{n-3} using the recurrence formula resulting to $P_{n-4}P_{n-5}$ $P_{n-5}P_{n-6}$.
- Keep expanding as above until P_{m+1} is expanded.

In order to identify positions of Padovan subwords in a Padovan word we prove the following two lemmas.

Lemma 6.3.8 *There is no occurrence of P_{n-1} in P_n, where $n \geq 1$.*

Proof It is easy to observe that the statement holds for $1 \leq n \leq 6$. For $n \geq 7$ we consider the following expansion of P_n:

$$P_n = P_{n-4}P_{n-5}P_{n-3}$$

Suppose there is an occurrence of P_{n-1} in P_n. Then it should be of form $xP_{n-5}y$, where x is a proper suffix of P_{n-4} and y is a prefix of P_{n-3}. This means x and y are borders of P_{n-1}, as $P_{n-1} = P_{n-3}P_{n-4}$. Theorem 6.3.5 shows that $2P_{n-7} \geq |x| + |y| = |P_{n-1}| - |P_{n-5}| = P_{n-4} + P_{n-6} = 2P_{n-6} + P_{n-7}$, which is impossible. □

Lemma 6.3.9 *P_{n-2} occurs in P_n only as a prefix of it, where $n \geq 3$.*

Proof It is easy to observe that the statement holds for $3 \leq n \leq 10$. For $n \geq 11$ we consider the following expansion of P_n:

$$P_n = P_{n-2}P_{n-5}P_{n-6}$$

Suppose there is another occurrence of P_{n-2} in P_n. Then it should be of form:

- xy, where x is a suffix of P_{n-2} and y is a non-empty prefix of P_{n-5}. This means x is a border of P_{n-2} and y is a border of P_{n-5}, as $P_{n-2} = P_{n-4}P_{n-5}$. Theorem 6.3.5 shows that $P_{n-8} + P_{n-11} \geq |x| + |y| = |P_{n-2}| = P_{n-4} + P_{n-5}$, which is impossible..
- $xP_{n-5}y$, where x is a suffix of P_{n-2} and y is a prefix of P_{n-6}. This means x and y are borders of P_{n-2}. Theorem 6.3.5 shows that $2P_{n-8} \geq |x| + |y| = |P_{n-2}| - |P_{n-5}| = P_{n-4} = P_{n-6} + P_{n-7}$, which is impossible.

□

Lemma 6.3.10 *The starting positions of the occurrences of P_m in P_n are the starting positions of P_m and $P_{m-2}P_{m-1}$ considered in the P_m, P_{m-1}, P_{m-2} expansion of P_n, where $n \geq m$ and $m \in \{4, \ldots, n-2\}$. In the case that $m \in \{2, 3\}$ the starting positions of the occurrences of P_m in P_n are the starting positions of P_m considered in the P_m, P_{m-1}, P_{m-2} expansion of P_n.*

Proof Using the recurrence relation we can get the P_m, P_{m-1}, P_{m-2} expansion of P_n as shown before:

$$P_n = \begin{cases} P_m P_{m-1} P_{m-1} P_{m-2} P_{m-1} P_{m-2} P_m \ldots, & m \equiv n \bmod 2 \\ P_{m-1} P_{m-2} P_m P_m P_{m-1} P_m P_{m-1} P_{m-1} \ldots, & m \not\equiv n \bmod 2 \end{cases} \tag{6.8}$$

We can now observe the occurrences of P_m in P_n mentioned in the beginning of this lemma. Any other occurrence should have one of the following forms (by Lemmas 6.3.8 and 6.3.9 and the fact that before any P_{m-2} in the above expansion there is always a P_{m-1} , as P_{m-2} is the result of the expansion of P_{m+1}):

- xy, where x is a non-empty suffix of P_m and y a non-empty prefix of P_m. Then both x and y are borders of P_m. Theorem 6.3.5 shows that $2P_{m-6} \geq |x| + |y| = |P_m| = P_{m-2} + P_{m-3}$, which is impossible.

- xy, where x is a non-empty suffix of P_m and y a non-empty prefix of P_{m-1}. Then x is a border of P_m. Theorem 6.3.5 shows that $|P_{m-6}| + |P_{m-1}| = |P_{m-6}| + |P_{m-3}| + |P_{m-4}| \geq |x| + |y| = |P_m| = |P_{m-2}| + |P_{m-3}| = |P_{m-3}| + |P_{m-4}| + |P_{m-5}|$ and therefore that $|P_{m-5}| = |P_{m-6}|$. This implies that $m \in \{6, 7, 8, 9\}$ which gives P_m not of form $P_{m-6}y$.

- xy, where x is a non-empty suffix of P_{m-1} and y a non-empty prefix of P_m. Then y is a border of P_m. Theorem 6.3.5 shows that $|P_{m-6}| + |P_{m-1}| = |P_{m-6}| + |P_{m-3}| + |P_{m-4}| \geq |x| + |y| = |P_m| = |P_{m-2}| + |P_{m-3}| = |P_{m-3}| + |P_{m-4}| + |P_{m-5}|$ and therefore that $|P_{m-5}| = |P_{m-6}|$. This implies that $m \in \{6, 7, 8, 9\}$ which gives P_m not of form xP_{m-6}.

- xy, where x is a non-empty suffix of P_{m-1} and y a non-empty prefix of P_{m-1}. As $P_{m-1}P_{m-1} = P_{m-3}P_{m-4}P_{m-3}P_{m-4}$ the occurrence takes one of the following forms:

  1. $zP_{m-4}w$, where z is a suffix of P_{m-3} and w is a non-empty prefix of P_{m-3}. Then w is a border of P_{m-3}. Theorem 6.3.5 shows that $|P_{m-3}| + |P_{m-9}| \geq |z| + |w| = |P_{m-3}| + |P_{m-5}|$, which is impossible.

  2. $zP_{m-4}P_{m-3}w$, where z is a suffix of P_{m-3} and w is a non-empty prefix of P_{m-4}. As P_{m-3} is a suffix of P_m there is an occurrence of P_{m-3} of form qw, where q is a suffix of P_{m-3}. Then q is a border of P_{m-3} and w is a border of P_m. Theorem 6.3.5 shows that $|P_{m-9}| + |P_{m-6}| \geq |q| + |w| = |P_{m-3}| = |P_{m-5}| + |P_{m-6}|$ and so $|P_{m-9}| \geq |P_{m-5}|$, which is impossible.

3. $zP_{m-3}w$, where z is a suffix of P_{m-4} and w is a non-empty prefix of P_{m-4}. Then z is a border of P_{m-4} and w is a border of P_m. Theorem 6.3.5 shows that $|P_{m-10}|+|P_{m-6}| \geq |z|+|w| = |P_{m-2}| = |P_{m-4}| + |P_{m-5}|$, which is impossible.

- xy, where x is a non-empty suffix of P_{m-1} and y a non-empty prefix of P_{m-2}. Then y is a border of P_m. Theorem 6.3.5 shows that there is no such case (situation is similar to the 3rd case).

- xy, where x is a non-empty suffix of P_{m-2} and y a non-empty prefix of P_m. Then x is a border of P_{m-2} and y is a border of P_m. Theorem 6.3.5 shows that $|P_{m-8}| + |P_{m-6}| \geq |x| + |y| = |P_m| = |P_{m-2}| + |P_{m-3}|$, which is impossible.

- xy, where x is a non-empty suffix of P_{m-2} and y a non-empty prefix of P_{m-1}. Then x is a border of P_{m-2}. Theorem 6.3.5 shows that $|P_{m-8}| + |P_{m-1}| \geq |x| + |y| = |P_m|$ and so $|P_{m-8}| \geq |P_{m-5}|$. This implies that $m = 8$ which gives P_m not of the form $P_{m-8}y$.

□

6.3.4 Covers of Padovan words

In this section we prove that a Padovan word has no covers except itself by proving that its longest border (given by Lemma 6.3.4) is not a cover of it (Lemma 6.3.11) and then applying Lemma 6.3.2.

Lemma 6.3.11 *P_{n-6} is not a cover of P_n, where $n \geq 14$.*

Proof Consider the $P_{n-5}, P_{n-6}, P_{n-7}$ expansion of P_n:

$P_n = P_{n-6}P_{n-7}P_{n-5}P_{n-5}P_{n-6}$

Expanding further the last P_{n-5} gives:

$P_n = P_{n-6}P_{n-7}P_{n-5}P_{n-11}P_{n-12}P_{n-10}P_{n-8}P_{n-6}$

To cover the position $|P_n| - |P_{n-6}|$ in P_n with an occurrence of P_{n-6} we need P_{n-6} to be a substring of the form:

- xy, where x is a non-empty suffix of P_{n-8} and y is a prefix of P_{n-6}. Then x is a border of P_{n-8} and y is a border of P_{n-6} with $|x| + |y| = |P_{n-6}|$. As of Lemma 6.3.5 there is no such case.

- $xP_{n-8}y$, where x is a non-empty suffix of P_{n-10} and y is a prefix of P_{n-6}.Then x is a border of P_{n-10} and y is a border of P_{n-6} with $|x| + |y| = |P_{n-9}|$. As of Lemma 6.3.5 there is no such case.

- $xP_{n-10}P_{n-8}y$, where x is a non-empty suffix of P_{n-12} and y is a prefix of P_{n-6}.Then x is a border of P_{n-12} and y is a border of P_{n-6} with $|x| + |y| = |P_{n-14}|$. As of Lemma 6.3.5 there is no such case.

□

Theorem 6.3.12 *P_n is the only cover of P_n, where $n \geq 0$.*

Proof With the help of Theorem 6.3.5 it is easy to observe that the theorem holds for $n \leq 13$. For $n \geq 14$ the theorem is a direct consequence of Lemma 6.3.11 and Lemma 6.3.2. □

6.3.5 Powers in Padovan words

In this section we identify those Padovan subwords that are also squares in another Padovan word by using the recursive formula and Lemma 6.3.10. We also observe that there are no Padovan words that are also cubes in P_n.

Theorem 6.3.13 *$P^2_{n-5}, P^2_{n-7}, P^2_{n-8} \ldots, P^2_1$ are the only squares of Padovan words that occur in P_n, where $n \geq 0$.*

Proof It is easy to observe that the statement holds for $0 \leq n \leq 7$. For $n \geq 8$ we will first show that the squares of the Padovan words not

considered above (namely P_{n-1}, P_{n-2}, P_{n-3}, P_{n-4}, P_{n-6} and P_{n-0}) are not squares in P_n. P_{n-1}^2 and P_{n-2}^2 are not squares in P_n (see Lemma 6.3.8 and Lemma 6.3.9). By Lemma 6.3.10, there is only one occurrence of P_{n-3} in P_n ($P_n = P_{n-2}P_{n-3}$) and so P_{n-3^2} is not a square in P_n. Similarly by Lemma 6.3.10, there is only one occurrence of P_{n-4} in P_n ($P_n = P_{n-4}P_{n-5}P_{n-3}$) and so P_{n-4}^2 is not a square in P_n. Furthermore Lemma 6.3.10 shows that there are three occurrences of P_{n-6} in P_n ($P_n = P_{n-6}P_{n-7}P_{n-7}P_{n-8}P_{n-7}P_{n-8}P_{n-6}$) that are not consecutive and so P_{n-6}^2 is not a square in P_n. Having a look at the definition of the Padovan word via the given morphism shows that in every Padovan word $c = P_0$ follows a b and thus no P_0P_0 occurs in any Padovan word.

We will show now that the Padovan words considered in the beginning of this theorem are squares of P_n. In order to achieve that we consider the following expansion of P_n:

$$P_n = P_{n-4}P_{n-5}P_{n-5}P_{n-6}$$

This reveals that P_{n-5}^2 is a square of P_n. P_n contains $P_{n-2}, P_{n-3}, \ldots, P_6$ as subwords and hence contains the above set of squares. □

Theorem 6.3.14 *There are no cubes of Padovan words that occur in P_n, where $n \geq 0$.*

Proof It is easy to observe that the statement holds for $0 \leq n \leq 3$. For $n \geq 4$ we consider the P_m, P_{m-1}, P_{m-2} expansion of P_n, where $m \in \{2, 3 \ldots, n-2\}$. Any P_m comes from either an expanded P_{m+3} and therefore it follows a $P_{m-1}P_{m-2} = P_{m+1}$ or from an expanded P_{m+2} and it is therefore followed by a P_{m-1}. By using similar arguments any $P_{m-2}P_{m-1}$ follows a P_{m-1} or and it is followed by a P_{m-2}, i.e. forces the appearance of $P_{m-1}P_{m-2}P_{m-1}P_{m-2} = P_{m-1}P_mP_{m-4}P_{m-2}$. Obviously there the P_m above does not follow a P_m as P_m and P_{m-1} have different

endings. Lemma 6.3.9 shows that the above P_m is not followed by a P_m. Occurrence of a $P_mP_mP_m$ in the word would appear from the first two cases but this violates at least one of the above statements as P_{m-2} is no suffix of P_m and P_{m-1} is no prefix of P_m, leading to a contradiction. It is left to check $P_0P_0P_0$, $P_1P_1P_1$, $P_{n-1}P_{n-1}P_{n-1}$. Theorem 6.3.13 suggests that $P_0P_0P_0$ does not occur in P_n. Having a look at the definition of the Padovan word via the given morphism shows that $P_1P_1P_1 = bbb$ appears via the action of the morphism on *ccc* or *acc*. However Theorem 6.3.13 shows that there is no *cc* in any Padovan word. Lemma 6.3.8 shows that $P_{n-1}P_{n-1}P_{n-1}$ does not occur in P_n. □

6.3.6 Conclusion and Future Work

Our main tool in proving the results of this article is Lemma 6.3.10 which gives an expansion of P_n in three smaller consecutive Padovan words. This expansion may not always be optimal but it provides a good framework to work on. Furthermore the following Lemma could have been useful to provide shorter proofs in some cases:

Lemma 6.3.15 *P_m occurs in P_n only as the image of P_{m-1}, where $n \geq 5$ and $5 \leq m \leq n$.*

Proof Any $P_5 = bca$ in P_n can only be the image of $P_4 = ab$. Any $P_6 = abbc$ in P_n can only be the image of $P_5 = bca$. Any $P_7 = bcaab$ in P_n is either the image of $P_6 = abbc$ or comes from a *abba*, which does not occur in any Padovan word (easy to prove by induction). For $m \geq 8$, any P_m in P_n can be expanded as $P_{m-2}P_{m-3}$ and by induction $P_m = f(P_{m-3})f(P_{m-4}) = f(P_{m-3}P_{m-4}) = f(P_{m-1})$ and therefore P_m occurs in P_n only as the image of P_{m-1}. □

In this section, inspired from the use of modified Padovan words (arising from a generalization of the Fibonacci sequence) to provide the cur-

rent lower bound for the maximum number of runs in a word, we gave the first formal study of Padovan words. We have defined them formally, we have identified all their borders and covers and we have revealed some interesting properties regarding their factor structure, their squares and cubes.

Beyond their obvious theoretical interest, those results might prove useful in testing algorithms that find periodicities or quasiperiodicities in strings and giving worst case examples on them. After some modifications one could be also able to provide good lower bounds on the number of periodicities and quasiperiodicities in words as in the paper by Simpson[220]. The above work could also be extended in general Sturmian words.

Chapter 7

String representations of trees

In this chapter we are investigating the use of words in attacking problems related to trees. More specifically we are investigating a problem related to tree pattern matching, a way of finding all repeating subtrees of a tree.

Tree pattern matching has been intensively studied over the past decades because of its various applications, among others, in mechanical theorem proving, term-rewriting, instruction selection, and non-procedural programming languages [146, 98, 170]. In addition, tree pattern matching has direct applications in computational biology, e.g. glycan classification [173], exact and approximate pattern matching and discovery in RNA secondary structure [193].

In many applications, it is essential to extract the repeated patterns in a tree within a mathematical structure [95, 116, 162]. In particular, the *common subtrees* problem consists of finding all of the subtrees having the same structure and the same labels on the corresponding nodes of two ordered labelled unranked trees [131]. This problem of equivalence, which is strictly related to the *common subexpression* problem [95, 116], arises, for instance, in the code optimisation phase of compiler design, or in saving storage for symbolic computations [5, 95, 116].

In this chapter, we consider a slightly different problem, and provide

a completely different solution to what has be done so far. We focus on finding all the repeating subtrees – the subtrees occurring more than once – in a tree structure. Notice that this problem can be solved by the algorithm presented in [131] in linear time. However, that solution requires the construction of a suffix tree, which is time consuming in practical terms. This problem is analogous to the well-known problem of finding all the repetitions in a given word [75]. Apart from its pleasing theoretical features, finding all the repeating subtrees, could be applied as an alternative solution, on the *maximum agreement subtree* (MAST) problem for trees representing the evolutionary history of a set of species [134]. That is, given a set of evolutionary leaf-labelled trees on the same set of taxa, the MAST problem consists of finding a subtree homeomorphically included in all input trees, and with the largest number of taxa. Furthermore our solution should prove useful in data compression (by using compact representation of trees) or tree pattern matching.

The proposed algorithm is divided into two phases: the preprocessing phase and the phase where all the repeating subtrees are computed. The preprocessing phase transforms the given tree to a string representing its postfix notation, and then computes arrays that store the height of each node of the tree, the parent of each node, and an indicator showing whether a node is the first child of its parent. The second phase, for computing all the repeating subtrees, is done in a bottom-up manner, using a partitioning procedure. The importance of the proposed algorithm is underlined by the fact that it can be applied in both unlabelled and labelled ordered ranked trees. Its linear runtime, as well as the use of linear auxiliary space, are important parts of its quality.

In what follows, we give some useful properties of trees in postfix notation. Subsection 7.1.2 describes the algorithm for finding all the repeating subtrees in unlabelled and labelled ordered ranked trees. Finally,

in Section 7.1.3 we present some experimental results [56, 55].

7.1 Subtree repeats

7.1.1 Preliminaries

In this subsection we present and prove some basic properties of trees in their postfix notation.

The following lemma shows that subtrees of a tree appear as substrings in its postfix notation.

Lemma 7.1.1 *Given a tree t and its postfix notation $post(t)$, the postfix notations of all subtrees of t are substrings of $post(t)$.*

Proof By induction on the height of the subtree:

1. If a subtree t' consists of only one node a ($|t'| = 1$ and $h(t') = 0$), where $\varphi(a) = 0$, then $post(t') = a$ and the claim holds for that subtree.

2. Assume the claim holds for subtrees $t_1, t_2, \ldots, t_p$, where $p \geq 1$ and $h(t_i) \leq m$, $1 \leq i \leq p$, $m \geq 0$. We must prove that the claim holds also for each subtree $t' = t_1 t_2 \ldots t_p a$, where $\varphi(a) = p$, $h(t') = m+1$: As $post(t') = post(t_1)\ post(t_2) \ldots post(t_p)\ a$, the claim holds for subtree t'.

□

The following lemma shows that the sum of arities of all nodes of a tree of order n is $n - 1$.

Lemma 7.1.2 *Given a ranked tree t of size n and its postfix notation $post(t) = x_1 x_2 \ldots x_n$, the sum of arities of all nodes of t is $\sum_{i=1}^{n} \varphi(x_i) = n - 1$.*

Proof The sum of arities of all nodes is the sum of the number of children of all nodes which is the sum of the edges hanging from each node. Each parent does not share children with any other node therefore each edge is counted only once. It is easy to see that there are $n-1$ edges in the tree (each node is hanging from an edge except the root of the tree). □

However, not every substring of a tree in postfix notation is a subtree in postfix notation. This is obvious due to the fact that for a given tree with n nodes in postfix notation, there can be $\mathcal{O}(n^2)$ distinct substrings but there are just n subtrees, each node of the tree is the root of one subtree. Just those substrings which themselves are trees in postfix notation are subtrees in postfix notation. This property is formalised by the following definition and lemma.

Definition Let $w = a_1 a_2 \dots a_m$, $m \geq 1$, be a string over a ranked alphabet $\mathcal{A}$. Then, the *arity checksum* $ac(w) = \varphi(a_1) + \varphi(a_2) + \ldots + \varphi(a_m) - m + 1 = \sum_{i=1}^{m} \varphi(a_i) - m + 1$.

Lemma 7.1.3 *Let $post(t)$ and w be a tree t in postfix notation and a substring of $post(t)$, respectively. Then, w is the postfix notation of a subtree of t, if and only if $ac(w) = 0$, and $ac(w_1) \geq 1$ for each w_1, where $w = xw_1$, $x \neq \varepsilon$.*

Proof For any two subtrees t_1 and t_2 it holds that $post(t_1)$ and $post(t_2)$ are either two different strings or one is a substring of the other. The former case occurs if the subtrees t_1 and t_2 are two different trees with no shared part and the latter case occurs if one tree is a subtree of the other tree. No partial overlapping of subtrees is possible in ranked ordered trees. Moreover, for any two adjacent subtrees it holds that their postfix notations are two adjacent substrings.

- *If:* By induction on the height of subtree t, where $w = post(t)$:

1. Assume that $h(t) = 1$, which means we consider the case $w = a$, where $\varphi(a) = 0$. Then, $ac(w) = 0$ and the claim holds for $h(t) = 1$.

2. Assume that the claim holds for subtrees $t_1, t_2, \ldots, t_p$ where $p \geq 1$, $h(t_1) \leq m$, $h(t_2) \leq m$, ..., $h(t_p) \leq m$, $ac(post(t_1)) = 0$, $ac(post(t_2)) = 0$, ..., $ac(post(t_p)) = 0$. We are to prove that it holds also for a subtree of height $m + 1$. Assume $w = post(t_1)post(t_2)$ $\ldots post(t_p)a$, where $\varphi(a) = p$. Then $ac(w) = ac(post(t_1)) + ac(post(t_2)) + \ldots + ac(post(t_p)) + p - (p + 1) + 1 = 0$ and $ac(w_1) \geq 1$ for each w_1 , where $w = xw_1$, $x \neq \varepsilon$.

   Thus, the claim holds for the case $h(t) = m + 1$.

- *Only if*: Assume $ac(w) = 0$, and $w = a_1a_2 \ldots a_k$, where $k \geq 1$, $Arity(a_k) = p$. Since $ac(w_1) \geq 1$ for each w_1, where $w = xw_1$, $x \neq \varepsilon$, none of the substrings w_1 can be a subtree in postfix notation. This means that the only possibility for $ac(w) = 0$ is that w is of the form $w = post(t_1)\ post(t_2) \ldots post(t_p)\ a$, where $p \geq 0$, and $t_1, t_2 \ldots t_p$ are adjacent subtrees. In such a case, $ac(w) = 0 + p - (p + 1) + 1 = 0$. No other possibility of the form of w for $ac(w) = 0$ is possible. Thus, the claim holds.

□

For the validity of our algorithm we also prove the following condition for a substring in postfix notation to be a subtree.

Lemma 7.1.4 *Let $post(t)$ and w be a tree t in postfix notation and a substring of $post(t)$, respectively. Then $w = w_1 \ldots w_{|w|}$, $w_i \in \mathcal{A}$, is the postfix notation of a subtree of t iff:*

- *w is composed by one leaf, or*
- *$ac(w) = 0$, w_1 corresponds to a node which is both a leaf and a first child, $|w| \geq 2$*

Proof First part is obvious.Now for the second part we need to prove:
Direct: Let w be the postfix notation of a subtree of t. Then w_1 corresponds to both a leaf and a first child as (due to postorder traversal of the tree). Also $ac(w) = 0$ as of Lemma 7.1.2.
Reverse: Let w be a substring of $post(t)$, such that $ac(w) = 0$, w_1 is both a leaf and a first child, and $|w| \geq 2$. Substrings of $post(t)$ starting from w_1 might end:

- on a node z whose subtree has w_1 as a leftmost leaf

  Then that substring is the postfix notation of the subtree rooted at z (we consider postorder traversal of the tree) and by Lemma 7.1.2 the arity checksum of that subtree is 0.

- on a node z whose subtree has a leftmost leaf found before w_1 in $post(t)$

  Then that substring is the postfix notation of the subtree rooted at z, which contains the largest subtree, say s, having w_1 as a leftmost leaf. However the siblings of s are missing and it is obvious to see that the sum of arities is greater that the size of the substring minus 1 (more children than the number of nodes minus 1) and so the arity checksum of that substring is greater than 0.

- on a node z whose subtree has a leftmost leaf found after w_1 in $post(t)$

  Then that substring contains a subtree, say s, having w_1 as a leftmost leaf and a collection of leaves and subtrees found later in the postorder traversal of the tree. It is obvious to see that the sum of

arities is less that the size of the substring minus 1 (less children than the number of nodes minus 1) and so the arity checksum of that substring is less than 0.

□

7.1.2 Algorithms

We consider the following problems:

Problem 6 *Find the complete subtree repeats of an unlabelled ordered ranked tree t consisting of n nodes.*

Problem 7 *Find the complete subtree repeats of a labelled ordered ranked tree t consisted of n nodes.*

In this section, we present algorithms solving Problems 6 and 7. The algorithms are divided in two phases: the preprocessing phase and the searching phase.

We divide the rest of this section in three subsections: first, we present the preprocessing part of the algorithm; then, we show a method for solving Problem 6 which we then extend to solve Problem 7.

Preprocessing

Given a tree t, its postfix notation is first computed using a simple postorder traversal of the tree.

While not necessary in general, a new identifier can be encoded for each node of the subject tree, based on its label and rank. These identifiers, along with the arity of the respective nodes, form the ranked alphabet. In this way, the case that the tree consists of nodes having the same label but different arity, can be easily handled.

Let $post(t) = x_1x_2 \dots x_n$ be the postfix notation of tree t. The preprocessing phase completes by computing 3 auxiliary arrays, which will be used during the searching phase:

1. The height of each subtree of t, having node x_i as its root, is stored in array H, for $1 \leq i \leq n$.

2. An array P of n elements, with the *i-th* element having the value p if x_p is the parent of x_i.

3. A binary array FC consisting of 1s and 0s, where the *i-th* element is set to 1 in case x_i is the first (leftmost) child of its parent node $x_{P[i]}$.

Finding subtree repeats

ALGORITHM SUBTREE-REPEATS($post(t) = x_1x_2 \dots x_n$ over ranked alphabet $\mathcal{A} = (\Sigma, \varphi)$)
1: $sc \leftarrow 1$;
2: **for** $i \leftarrow 1$ to n **do**
3: **if** $\varphi(x_i) = 0$ **then**
4: $S \leftarrow S \bigcup \{i\}$;
5: $T[i] \leftarrow sc$;
6: $TL[i] \leftarrow 1$;
7: **else**
8: $T[i] \leftarrow 0$;
9: $TL[i] \leftarrow 0$;
10: **for** $i \leftarrow 1$ to $H[n]$ **do**
11: **while** LA[i] is not empty **do**
12: PARTITION(DEQUEUE($LA[i]$))
13: **return** Sets of starting positions of substrings of $post(t)$ and their lengths, representing subtrees from t;

We are now in a position to present Algorithm SUBTREE-REPEATS, solving Problem 6. The computation is based on a bottom-up traversal of the input tree t, described by its postfix notation $post(t) = x_1x_2 \dots x_n$. The algorithm, at each step (level) i, locates and outputs all repeating subtrees of height i. We also introduce an auxiliary array called *level*

array (LA), which keeps track of queues of triplets. These triplets describe factors of $post(t)$, and are of the form (S, ℓ, ac), where S is a set containing the starting positions of the occurrences of the factor, ℓ is the factor's length, and ac is its arity checksum, which, in case is 0, indicates that the factor corresponds to some subtree of t.

The algorithm starts by constructing a triplet $(S, 1, 0)$, representing all leaves, i.e. subtrees of height 0 (lines 3-6), with its set S containing all positions of the unary symbol in $post(t)$. The triplet is then passed to the function ASSIGN-LEVEL, which splits the elements of S in several subsets, according to the height of the subtree specified by the parent (stored in $H[P[root]]$) of each element in S (line 4). Elements which do not correspond to subtrees being leftmost children (first children) of their parent nodes are discarded. The resulting subsets are then wrapped into triplets and appended in the appropriates queues of LA according to $H[P[root]]$. Note, that function ASSIGN-LEVEL takes as input only triplets describing factors that correspond to subtrees.

At each step i of the algorithm, the triplets in $LA[i]$ are passed to the function PARTITION, which partitions a triplet according to the next factor, starting at position r (line 2), that is to be concatenated with the factor described by the triplet. The next factor either represents a subtree of t in postfix notation, in case its first symbol marks the beginning of a subtree (lines 3-5), or a single symbol (lines 6-8). Triplets are recursively partitioned (line 18) until they describe factors representing subtrees in postfix notation (lines 10-16). When a triplet finally describes a factor representing a subtree, starting positions of those factors are assigned an index (stored in array T), and their lengths are stored in an array TL (lines 14-15). This is to indicate that the factor starting at position i i having length $TL[i]$ was found to correspond to a subtree. Those triplets are then processed by ASSIGN-LEVEL, which partitions them in

levels and stores them in the appropriate queues of LA. As stated before, only the elements i of the set S of a triplet $E = (S, \ell, 0)$, such that the subtree corresponding to the factor $x_i \dots x_{i+\ell}$ is the first children of the subtree specified by its parent node, are considered (Lemma 7.1.4). The algorithm terminates when the level array LA is empty.

ALGORITHM ASSIGN-LEVEL($E = (S, \ell, ac)$)
1: **for** $i \in S$ **do**
2: $root = i + TL[i] - 1$;
3: **if** $FC[root] = 1$ **then**
4: $S_{H[P[root]]} \leftarrow S_{H[P[root]]} \cup \{i\}$;
5: $L \leftarrow L \cup H[P[root]]$;
6: **for** $i \in L$ **do**
7: ENQUEUE($LA[i], (S_i, \ell, 0)$);
8: **return** Partitioning of E in levels;

ALGORITHM PARTITION($E = (S, \ell, ac)$, $post(t) = x_1 x_2 \dots x_n$)
1: **for** $i \in S$ **do**
2: $r = i + \ell$;
3: **if** $T[r] \neq 0$ **then**
4: $E_{T[r]} \leftarrow (S_{T[r]} \cup \{i\}, \ell + TL[r], ac - 1)$;
5: $L \leftarrow L \cup \{E_{T[r]}\}$;
6: **else**
7: $E_{x_r} \leftarrow (S_{x_r} \cup \{i\}, \ell + 1, ac - 1 + \varphi(x_r))$;
8: $L \leftarrow L \cup \{E_{x_r}\}$;
9: **for** $E_i = (S_i, \ell_i, ac_i) \in L$ **do**
10: **if** $ac_i = 0$ **then**
11: OUTPUT(S_i, ℓ_i);
12: $sc = sc + 1$;
13: **for** $j \in S_i$ **do**
14: $T[j] \leftarrow sc$;
15: $TL[j] \leftarrow \ell_i$;
16: ASSIGN-LEVEL(E_i);
17: **else**
18: PARTITION(E_i);
19: **return** Partitioning of E in classes corresponding to next element to be considered;

Theorem 7.1.5 *The algorithm* SUBTREE-REPEATS *computes all subtree repeats of a given tree t in $\Theta(n)$ time, where $|t| = n$.*

Proof The preprocessing phase, i.e. the computation of $post(t)$, arrays P,H and FC, is done in $\Theta(n)$ time. During the expansion of the subtrees performed in the function PARTITION, the algorithm does not read a symbol more than once, but rather reads the previously expanded subtrees. Merging the subtrees is done in $n-1$ operations (number of children of the tree). □

Solving the problem for labelled ordered ranked trees

The algorithm described in the previous section can be adapted to solve Problem 7 using a slight modification, which we present in this section.

During initialization we should pay attention to form different sets of leaves according to their label. Additionally, partitioning should be done according to the label, rank and subtree of the next element that is to be considered. In that way we will require a $(\Sigma+1)n \times 1$ array of sets to be formed in order to obtain the partitioning. However we can do better than this by separating the partitioning in the following cases:

- When we partition with respect to subtree starting from next element to be considered we use an auxiliary $n \times 1$ array of sets.
- When we partition with respect to next element to be considered we first use an auxiliary $n \times 1$ array of sets to partition with respect to the arity of the element and then an auxiliary $\Sigma \times 1$ array of queues to partition with respect to the label of the element. We can restrict to the different labels that appear in the tree, say k (clearly $k \leq n$), thus using an auxiliary $k \times 1$ array of queues at the second step of the partitioning, thus using $\Theta(n)$ space during the execution of the algorithm.

The procedures that differ from the previous algorithm are given in the Appendix.

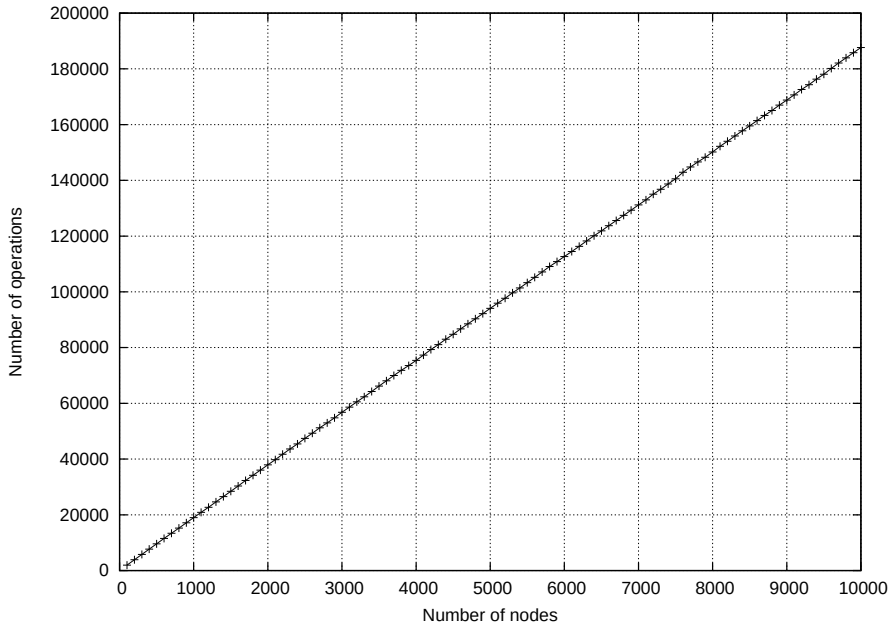


Figure 7.1: Number of operations performed by Algorithm SUBTREE-REPEATS, according to the size of the input tree

7.1.3 Experiments

We have conducted an experiment to verify the linear runtime of the proposed algorithm computing subtree repeats in unlabelled ordered ranked trees, in practice. For the experiment we have used randomly generated trees with their size ranging from 100 to 10000 nodes with a step of 100 nodes and a bounded alphabet with the arity of the symbols ranging between 0 and 5. Figure 7.1 shows the number of operations carried out by our implementation of the algorithm against the number of nodes of the tree instances. The resulting graph clearly indicates the linear relationship between the runtime and the number of nodes of tree instances.

Part IV

Graphs

Chapter 8

Extremal graph theory

Graphs arise in many areas of mathematics and computer science having applications in many other fields as well. Extremal graph theory problems usually ask for the maximum or minimum size or order of a graph having certain characteristics (see also [29]). Such questions are often quite natural in the construction of networks or circuits.

We consider variants of the following problems:

- Degree/diameter problem: What is the maximum number of vertices $n_{d,k}$ that can be contained in a graph of maximum degree d and diameter at most k? (The order of such a graph is between 2 (a P_2) and the Moore bound [145].)

- EX-problem: Given a graph of order n what is the maximum number of edges, denoted by $ex(n;t)$, that can exist in the graph such that it contains no cycle C_k, where $3 \leq k \leq t$?

- Ramsey [206] showed that in a blue-red colouring of the edges of a sufficiently large complete graph (K_n) there must exist either a blue or a red complete subgraph of a given order (a blue K_m or a red K_m). The minimum order (n) of a complete graph that must achieve that is known as a Ramsey number.

- Degree/girth problem: Given natural numbers $d \geq 2$ and $g \geq 3$, find the smallest possible number $n(d, g)$ of vertices in a regular graph of degree d and girth g. A first proof for the existence of such graphs was given by Sachs [216]. That upper bound was further improved by him in a collaboration with Erdős a few months later [107].

- Zarankiewicz problem: what is the maximum number of edges that can exist in a bipartite graph G, whose disjoint sets U and V have size m and n respectively, such that $K_{s,t}$ is not a subgraph of G?

8.1 Degree/diameter problem for trees and pseudotrees

In this section we consider the degree/diameter problem which asks: What is the maximum number of vertices $n_{d,k}$ that can be contained in a graph of maximum degree d and diameter at most k? Moore [145] initially posed the problem and introduced the upper bound called the *Moore bound*. Graphs that attain the *Moore bound* are called *Moore graphs*.

For most values of d and k, the degree/diameter problem is open; we only have general upper and lower bounds for the values of $n_{d,k}$. In an effort to tighten the gaps between the upper and lower bounds, research activities related to the degree/diameter problem fall into two main streams. On the one hand there are proofs of non-existence of graphs of order equal to the current best upper bounds, thereby improving (lowering) the upper bounds[19, 106, 175]. On the other hand, there is a great deal of activity in the constructions of large graphs, furnishing better lower bounds on $n_{d,k}$[40, 102, 195].

The study of Moore graphs was initiated by Hoffman and Singleton [145], their pioneering paper was devoted to Moore graphs of diameter 2 and 3. In the case of diameter 2, they proved that Moore graphs exist for $d \in \{2, 3, 7\}$ and possibly 57 but for no other degrees, and that for the first three values of d the graphs are unique. For diameter 3 they showed that the unique Moore graph is the heptagon (for $d = 2$). The proofs exploited eigenvalues and eigenvectors of the adjacency matrix (and its principal submatrices) of graphs.

It turns out that no Moore graphs exist for $d \geq 3$ and $k \geq 3$. This was shown by Damerell [90] by way of an application of his theory of distance-regular graphs to the classification of Moore graphs. An independent proof of this result was also given by Bannai and Ito[18].

The main results concerning Moore graphs can be summed up as follows. Moore graphs for diameter $k = 1$ and maximum degree $d \geq 1$ are the complete graphs K_{d+1}. For diameter $k = 2$, Moore graphs are the cycle C_5 for degree $d = 2$, the Petersen graph for degree $= 3$ and the Hoffman-Singleton graph for degree $d = 7$. The existence or otherwise of a Moore graph of degree 57 and diameter 2 is still unknown. Finally, for diameter $k \geq 3$ and degree $d = 2$, Moore graphs are the cycles on $2k + 1$ vertices C_{2k+1}.

Since the general degree/diameter problem is difficult, research has been also concentrated on various related problems. These include studies of the degree/diameter problem for special types of graphs such as Cayley[89, 238], planar[112, 124, 235], bipartite[20, 72], directed[21, 35, 163] and toroidal[234] graphs (for a general survey of the degree/diameter problem see [194]). Several other areas of research in graph theory turn out to be related or inspired by the theory of Moore graphs; examples include cages, antipodal graphs, Moore geometries and Moore groups. Recall that a $(k; g)$-*cage* is a graph of degree k and girth g, with the

minimum possible number of vertices. Connections between cages and Moore graphs are explained in a survey paper on cages by Wong [239].

In this section we consider the degree/diameter problem on structures that have not been explored before, trees, special types of trees such as Cayley trees, caterpillars, lobsters, banana trees and firecracker trees, as well as for tree-like structures such as pseudotrees, giving the extremal numbers and constructions[68].

8.1.1 Definitions and Problems

In this paper we consider the following problems.

Problem 8 *Given natural numbers $d \geq 1$ and $k \geq 1$ find the largest possible number of vertices $n_{d,k}$ in a tree of maximum degree d and diameter at most k. Within this problem we consider also the specific cases of Cayley trees, caterpillars, lobsters, banana trees and firecracker trees.*

Problem 9 *Given natural numbers $d \geq 2$ and $k \geq 1$ find the largest possible number of vertices $n_{d,k}$ in a pseudotree of maximum degree d and diameter at most k.*

8.1.2 Degree/diameter problem on trees

In this section we investigate the degree/diameter problem on trees, and in particular on some special types of trees such as Cayley trees, caterpillars, lobsters, banana trees and firecracker trees, as well as for tree-like structures such as pseudotrees, giving the extremal numbers and constructions. Unlike the general case when the Moore bound is usually not achieved, in the case of banana trees, caterpillars and lobsters, we can always construct the corresponding Moore graph as well as for trees, Cayley trees and pseudotrees in the case that k is even.

We attack the degree/diameter problem on a tree by considering a maximal shortest path and maximizing the order of hanging subtrees on it. Before going further we introduce the following helpful notation:

- $F_{d,h}$ is the tree where all non-leaf nodes have degree d and all its leaves are at height h from the root. ($|F_{d,h}| = \frac{d(d-1)^h-2}{d-2}$)
- $F'_{d,h}$ is the tree where all non-leaf nodes have degree d, except its root which has degree $d-1$, and all its leaves are at height h from the root. ($|F'_{d,h}| = \frac{(d-1)^{h+1}-1}{d-2}$)
- $F''_{d,h}$ is the tree where all non-leaf nodes have degree d, except its root which has degree $d-2$, and all its leaves are at height h from the root. ($|F''_{d,h}| = (d-1)^h$)

Theorem 8.1.1 *For a tree of maximum degree d and diameter at most k, we have*

$$n_{d,k} = \begin{cases} 2, & d = 1 \text{ or } k = 1 \\ k+1, & d = 2 \text{ and } k \geq 2 \\ \frac{2(d-1)^{\frac{k+1}{2}}-2}{d-2}, & k \text{ odd, } k \geq 3 \text{ and } d \geq 3 \\ \frac{d(d-1)^{\frac{k}{2}}-2}{d-2}, & k \text{ even, } k \geq 2 \text{ and } d \geq 3 \end{cases} \tag{8.1}$$

Proof For $d = 1$ or $k = 1$ we have $n_{1,k} = 2$ and $n_{d,1} = 2$, achieved by P_2.

For $d = 2$ and $k \geq 2$ we have $n_{2,k} = k + 1$, achieved by P_{k+1}.

For $d \geq 3$ and $k \geq 2$, we consider a maximal shortest path of length r.

We then number its vertices from 0 to r.

No subgraph hanging from any one of these vertices has common vertices with a subgraph hanging from another vertex as then we would have a cycle.

It is easy to see that these hanging subgraphs are also trees.

An end vertex has no hanging tree attached as that would mean a larger maximal shortest path.

The tree hanging from each non-end vertex i has height at most $min(i, r-i)$ (consider distance of the leaves from the end vertices 0 and r).

Therefore, there is at most one $F''_{d,min(i,r-i)}$ hanging from the vertex i.

Summing the vertices gives the required upper bound:

$\sum_{i=0}^{r} |F''_{d,min(i,r-i)}|$

Clearly the above is maximized when $r = k$.

For k odd,

$n_{d,k} = 2\sum_{i=0}^{\frac{k-1}{2}}(d-1)^i = 2\frac{(d-1)^{\frac{k+1}{2}}-1}{(d-1)-1} = \frac{2(d-1)^{\frac{k+1}{2}}-2}{d-2}$

For k even:

$n_{d,k} = |F''_{d,\frac{k}{2}}| + 2\sum_{i=0}^{\frac{k}{2}-1}|F''_{d,i}| = (d-1)^{\frac{k}{2}} + 2\sum_{i=0}^{\frac{k}{2}-1}(d-1)^i$

$= (d-1)^{\frac{k}{2}} + 2\frac{(d-1)^{\frac{k}{2}}-1}{(d-1)-1} = \frac{d(d-1)^{\frac{k}{2}}-2}{d-2}$ □

Observe that the extremal case for any general tree is a Cayley tree and therefore the upper bound for Cayley trees is the same as for general trees; it is in fact the (unique) solution of the degree/diameter problem for Cayley graphs. However, this is not the case for other types of trees. The next theorems deal with the degree/diameter problem for caterpillars, lobsters, banana trees and firecracker trees.

The extremal cases for caterpillars and lobsters are found by considering the limitations of the tree as in Theorem 8.1.1 and then applying the limitations for the specific tree. We give the extremal cases without proof.

Theorem 8.1.2 *For a caterpillar tree:* $n_{d,k} = 2 + (k-1)(d-1)$

Theorem 8.1.3 *For a lobster tree:*

$$n_{d,k} = \begin{cases} 2, & k = 1 \text{ or } d = 1 \\ d+1, & k = 2 \text{ and } d \geq 2 \\ (k-3)d^2 + (8-2k)d + k - 3 & k \geq 3 \text{ and } d \geq 2 \end{cases} \tag{8.2}$$

The extremal cases for banana and firecracker trees are found by considering the different type of constructions of those trees. Again we give the extremal cases without proof.

Theorem 8.1.4 *For a firecracker tree:*

$$n_{d,k} = \begin{cases} 2, & k = 1 \text{ or } d = 1 \\ d+1, & (k \in \{2,3\} \text{ and } d \geq 3) \text{ or } (k = 4 \text{ and } d \geq 5) \\ k+1, & d = 2 \text{ and } k \geq 2 \\ 2k-2, & k = 4 \text{ and } d \in \{3,4\} \\ (k-3)(d+1), & k \geq 5 \text{ and } d \geq 3 \end{cases} \tag{8.3}$$

Proof For the general case, there are four constructions to consider:

- A S_{d+1}
- $k+1$ copies of S_1 joined in a path
- $k-1$ copies of S_2 joined in a path
- $k-3$ copies of S_{d+1} joined in a path

When $d = 1$ or $k = 1$ the only possible graph is a P_2, which has diameter 1 (a S_2 or 2 copies of S_1 joined in a path). When $d = 2$ and $k \geq 2$ the second construction gives the extremal case. For $d \geq 3$, we are able to pick any of the above constructions (depending on k only). Hence:

- when ($k \in \{2,3\}$ and $d \geq 3$) or ($k = 4$ and $d \geq 5$), the first construction gives the extremal case.
- when $k = 4$ and $d \in \{3,4\}$, the third construction gives the extremal case.
- when $k \geq 5$ and $d \geq 3$, the fourth construction gives the extremal case.

Theorem 8.1.5 *For a banana tree:*

$$n_{d,k} = \begin{cases} 2, & k = 1 \text{ or } d = 1 \\ d+1, & k = 2 \\ d+2, & k = 3 \\ 2d+1, & k = \{4,5\} \\ 1+d+d^2, & k \geq 6 \end{cases} \tag{8.4}$$

Proof For the general case, there are four constructions to consider:

- a S_{d+1}, i.e. d copies of S_1 attached on the root
- a S_{d+1} attached on the root
- d copies of S_2 attached on the root
- d copies of S_{d+1} attached on the root

When $d = 1$ or $k = 1$, the only possible graph is a P_2, which has diameter 1 (a S_1 attached on the root). When $k = 2$, the first construction gives the extremal case. When $k = 3$, the second construction gives the extremal case. When $k \in \{4,5\}$, the third construction gives the extremal case. When $k \geq 6$, the fourth construction gives the extremal case.

Although a pseudotree is very close to being a tree, it appears that a deeper combinatorial analysis is needed to find the extremal cases of the degree/diameter problem on a pseudotree. To attack the problem we consider cases on the maximum height of subtrees hanging from the cycle of the pseudotree, thus deriving our results.

The following upper bound will be very helpful as it will allow us to disregard constructions with subtrees of small maximum height h on their cycle.

Lemma 8.1.6 *A pseudotree of maximum degree d, diameter at least k, having subtrees of maximum height h on its cycle and at least one subtree achieving that height has at most $2(k-2h)(d-1)^h + \frac{d(d-1)^h-2}{d-2}$ vertices, where $h \leq \lfloor \frac{k}{2} \rfloor$, $d \geq 3$ and $k \geq 2$.*

Proof Clearly all subtrees hanging from the cycle of the pseudotree must be of the form $F''_{d,q}$, where $0 \leq q \leq h$. On the right side of the subtree of height h there can be at most $k-2h$ subtrees of max height h, then the height of the subtrees gradually decreases as $h-1, h-2, \ldots, 1, 0$. Same on the left of the central subtree of height h. One can observe that

$n \leq 2(k-2h)|F''_{d,h}| + |F_{d,h}|$

$= 2(k-2h)(d-1)^h + \frac{d(d-1)^h-2}{d-2}$. □

The next lemma gives the extremal case for k even, $k \geq 2$, and $d \geq 4$ using the upper bound of Lemma 8.1.6.

Lemma 8.1.7 *When k is even, a pseudotree of maximum degree d and diameter at most k has at most the same number of nodes as the extremal tree of maximum degree d and diameter at most k, where $k \geq 2$ and $d \geq 4$ or $k \geq 4$ and $d = 3$.*

Proof Hanging subtrees are of the form $F''_{d,h}$ except possibly one with height $\lceil \frac{k}{2} \rceil$.

Clearly, on the right side of the subtree of height $\frac{k}{2}$ the height of the subtrees gradually decreases as $\frac{k}{2}-1, \frac{k}{2}-2, \ldots, 1, 0$. Same on the left side of the subtree of height $\frac{k}{2}$.

Observe that $n = F_{d,\frac{k}{2}}$.

Subtracting from that the upper bound of Lemma 8.1.6 we get:

$|F_{d,\frac{k}{2}}| - 2(k-2h)(d-1)^h - \frac{d(d-1)^h-2}{d-2}$

$= \frac{d(d-1)^{\frac{k}{2}}-2}{d-2} - 2(k-2h)(d-1)^h - \frac{d(d-1)^h-2}{d-2}$

$= \frac{(d-1)^h}{d-2}[d(d-1)^{\frac{k}{2}-h} - d - 2(k-2h)(d-2)]$

It is enough to prove $d(d-1)^{\frac{k}{2}-h} - d - 2(k-2h)(d-2) \geq 0$.

Letting $\frac{k}{2} - h = x$ transforms the above to $d(d-1)^x - d - 4(d-2)x \geq d((d-1)^x - 1 - 4x)$ so it is enough to show that $((d-1)^x - 1 - 4x) \geq 0$.

It is easy to see that this holds for all feasible values except for the pairs (d, x) = (3,1), (3,2), (3,3), (3,4), (4,1) and (5,1). Trying these pairs in $d(d-1)^x - d - 4(d-2)x$, only $(d = 3, x = 1)$ gives a negative value. We will next show that the pair $(d = 3, x = 1)$ does not give constructions with more nodes than the extremal tree of maximum degree 3 and diameter at most k, where k is even and $k \geq 4$.

We will consider cases on the number of hanging subtrees of height $\frac{k}{2}-1$. With a little combinatorial analysis we end up with the following cases:

- C_5 with 5 $F''_{d,\frac{k}{2}-1}$ attached to it.
- C_5 with 4 $F''_{d,\frac{k}{2}-1}$ and one $F''_{d,\frac{k}{2}-2}$ attached to it.
- C_7 with 3 $F''_{d,\frac{k}{2}-1}$ and 4 $F''_{d,\frac{k}{2}-2}$ attached to it.
- C_k with 2 $F''_{d,\frac{k}{2}-1}$ and 5 $F''_{d,\frac{k}{2}-2}$ attached to it.
- In the case that we are dealing with one such subtree we get as extremal a C_{k+1} with one $F''_{d,\frac{k}{2}-1}$ attached to it and a sequence of $F''_{d,\frac{k}{2}-2}$, $F''_{d,\frac{k}{2}-2}$, $F''_{d,\frac{k}{2}-2}$, $F''_{d,\frac{k}{2}-3}$, $F''_{d,\frac{k}{2}-4}$, ..., $F''_{d,2}$, $F''_{d,1}$, $F''_{d,0}$ both on its left and right sides (order is $|F_{d,\frac{k}{2}-1}| + 4|F''_{d,\frac{k}{2}-2}|$).

Clearly, the first case gives the maximum order.

Subtracting the order of the first case from the order of the extremal tree we get

$$|F_{3,\frac{k}{2}}| - 5|F''_{3,\frac{k}{2}-1}| = 3 \cdot 2^{\frac{k}{2}} - 2 - 5 \cdot 2^{\frac{k}{2}-1} = 2^{\frac{k}{2}-1} - 2 \geq 0. \qquad \square$$

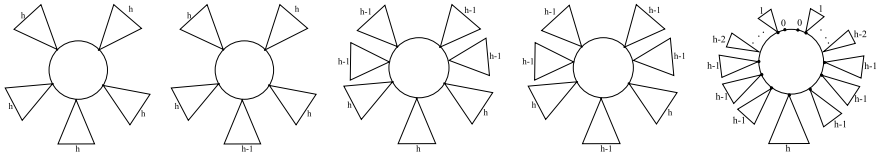


Figure 8.1: Cases considered in Lemma 8.1.7, a $\triangle$ with an h next to it stands for an $F''_{d,h}$ ($h = \frac{k}{2} - 1$).

The following lemma will be useful when considering the problem for k odd.

Lemma 8.1.8 *For k odd, a pseudotree of maximum degree $d \geq 4$, diameter at most k, having subtrees of maximum height $\frac{k-1}{2}$ on its cycle, and at least one subtree achieving that height, has at most $3|F''_{d,\frac{k-1}{2}}|$ nodes.*

Proof We consider cases according to the number of hanging subtrees of height $\frac{k-1}{2}$. With a little combinatorial analysis we arrive at the following cases:

- C_3 with 3 $F''_{d,\frac{k-1}{2}}$ attached to it.
- C_5 with 2 $F''_{d,\frac{k-1}{2}}$ and 3 $F''_{d,\frac{k-3}{2}}$ attached to it or a C_{k+2} with two consecutive $F''_{d,\frac{k-1}{2}}$ attached to it and a sequence of $F''_{d,\frac{k-3}{2}}$, $F''_{d,\frac{k-5}{2}}$, $\ldots$,$F''_{d,2}$, $F''_{d,1}$, $F''_{d,0}$ both on their left and right sides (order is $|F_{d,\frac{k-1}{2}}|$ $+|F''_{d,\frac{k-1}{2}}|$).
- In the case that we are dealing with one such subtree we get as an upper bound a C_{k+2} with consecutive $F''_{d,\frac{k-3}{2}}$, $F''_{d,\frac{k-1}{2}}$, $F''_{d,\frac{k-3}{2}}$ attached to it and a sequence of $F''_{d,\frac{k-3}{2}}$, $F''_{d,\frac{k-5}{2}}$, $\ldots$,$F''_{d,2}$, $F''_{d,1}$, $F''_{d,0}$ both on their left and right sides (order is $|F_{d,\frac{k-1}{2}}| + 2|F''_{d,\frac{k-3}{2}}|$).

Clearly, the first case gives the maximum order. □

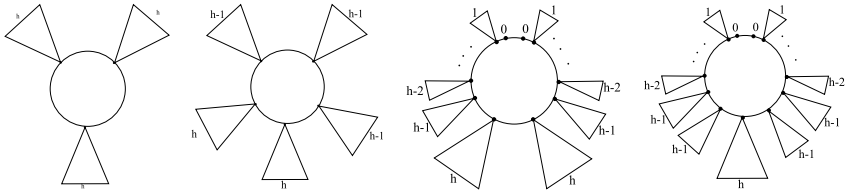


Figure 8.2: Cases considered in Lemma 8.1.8 a $\triangle$ with an h next to it stands for an $F''_{d,h}$ $(h = \frac{k-1}{2})$

Using the upper bound of Lemma 8.1.6, the next lemma gives the extremal case for k odd, $k \geq 3$, and $d \geq 4$.

Lemma 8.1.9 *When k is odd, a pseudotree of maximum degree d and diameter at most k has at most $3|F''_{d,\frac{k-1}{2}}|$ nodes, where $d \geq 4$ and $k \geq 3$.*

Proof Subtracting the extremal tree of height $\frac{k+1}{2}$ we get:

$3|F''_{d,\frac{k}{2}-1}| - \frac{2(d-1)^{\frac{k+1}{2}}-2}{d-2}$

$= 3(d-1)^{\frac{k-1}{2}} - \frac{2(d-1)^{\frac{k+1}{2}}-2}{d-2}$

$= \frac{3(d-1)^{\frac{k-1}{2}}(d-2)+2-2(d-1)^{\frac{k+1}{2}}}{d-2} = \frac{(d-1)^{\frac{k-1}{2}}(3d-10)+2}{d-2} > 0$

Subtracting the upper bound of Lemma 8.1.6 we get:

$3|F''_{d,\frac{k}{2}-1}| - 2(k-2h)(d-1)^h + \frac{d(d-1)^h-2}{d-2} = 3(d-1)^{\frac{k-1}{2}} - 2(k-2h)(d-1)^h + \frac{d(d-1)^h-2}{d-2}$

$> (d-1)^h[3(d-1)^{\frac{k-1}{2}-h} - 2(k-2h) - \frac{d}{d-2}]$

Consider:

$3(d-1)^{\frac{k-1}{2}-h} - 2(k-2h) - \frac{d}{d-2} \geq 3(d-1)^{\frac{k-1}{2}-h} - 2(k-2h) - 3$

Substituting $x = \frac{k-1}{2} - h$ the above expression becomes $3(d-1)^x - 4x - 5$.

It suffices to prove that $3(d-1)^x - 4x - 5 \geq 0$.

It is easy to see that this holds for all valid values except for the pairs $(d = 3, x = 1)$ and $(d = 3, x = 2)$. □

The following lemma proves useful when considering the problem for k odd, $d = 3$ and $k \geq 5$.

Lemma 8.1.10 *For k odd, a pseudotree of maximum degree 3, diameter at least k, having subtrees of maximum height $\frac{k-3}{2}$ on its cycle and at least one subtree having that height, has at most $7|F''_{3,\frac{k-3}{2}}|$ nodes.*

Proof We consider cases according to the number of hanging subtrees of height $\frac{k-3}{2}$.
With a little combinatorial analysis we identify all the possible cases as follows.

- C_7 with 7 $F''_{3,\frac{k-3}{2}}$ attached to it.
- C_7 with 6 $F''_{3,\frac{k-3}{2}}$ and one $F''_{3,\frac{k-5}{2}}$ attached to it.
- C_7 with 5 $F''_{3,\frac{k-3}{2}}$ and 2 $F''_{3,\frac{k-5}{2}}$ attached to it.
- C_9 with 4 $F''_{3,\frac{k-3}{2}}$ and 5 $F''_{3,\frac{k-5}{2}}$ attached to it.
- C_9 with 3 $F''_{3,\frac{k-3}{2}}$ and 6 $F''_{3,\frac{k-5}{2}}$ attached to it.
- In the case that we are dealing with 2 such subtrees we get as an upper bound the construction for the case below with one of $F''_{3,\frac{k-5}{2}}$ at a distance at most 3 from the central $F''_{3,\frac{k-3}{2}}$ replaced by one $F''_{3,\frac{k-3}{2}}$ (order is $|F_{3,\frac{k-3}{2}}| + 5|F''_{3,\frac{k-5}{2}}| + |F''_{3,\frac{k-3}{2}}|$).
- In the case that we are dealing with one such subtree we get as an upper bound a C_{k+2} with consecutive $F''_{3,\frac{k-5}{2}}$, $F''_{3,\frac{k-5}{2}}$, $F''_{3,\frac{k-5}{2}}$, $F''_{3,\frac{k-3}{2}}$, $F''_{3,\frac{k-5}{2}}$, $F''_{3,\frac{k-5}{2}}$, $F''_{3,\frac{k-5}{2}}$ attached to it and a sequence of $F''_{3,\frac{k-5}{2}}$, $F''_{3,\frac{k-7}{2}}$, ..., $F''_{3,2}$, $F''_{3,1}$, $F''_{3,0}$ both on its left and right sides (order is $|F_{3,\frac{k-3}{2}}| + 6|F''_{3,\frac{k-5}{2}}|$).

Clearly, the first case gives the maximum order. □

Using Lemmas 8.1.9 and 8.1.10, the following lemma gives the extremal case for k odd, $k \geq 5$, and $d = 3$.

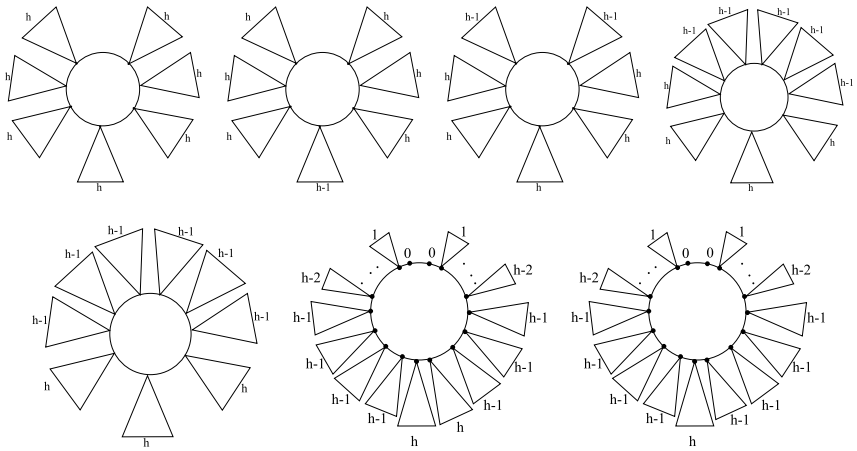


Figure 8.3: Cases considered in Lemma 8.1.10, a $\triangle$ with an h next to it stands for an $F''_{d,h}$ ($h = \frac{k-3}{2}$.)

Lemma 8.1.11 *For k odd, $k \geq 3$, a pseudotree of maximum degree 3 and diameter at most k has at most the same number of nodes as the corresponding extremal tree.*

Proof The extremal construction with maximum height hanging subtree $F''_{3,\frac{k-1}{2}}$ (Lemma 8.1.8) gives the same order since:
$(2 \cdot 2^{\frac{k+1}{2}} - 2) - (|F_{\frac{k-1}{2}}| + |F''_{\frac{k-1}{2}}|) = 4 \cdot 2^{\frac{k-1}{2}} - 2 - 3 \cdot 2^{\frac{k-1}{2}} + 2 - 2^{\frac{k-1}{2}} = 0$
Similarly, the extremal construction with maximum height hanging subtree $F''_{3,\frac{k-3}{2}}$ gives:
$(2 \cdot 2^{\frac{k+1}{2}} - 2) - 7|F''_{3,\frac{k-3}{2}}| = (2 \cdot 2^{\frac{k+1}{2}} - 2) - 7 \cdot 2^{\frac{k-3}{2}}$
$= 2^{\frac{k-3}{2}} - 2 \geq 0$
Upper bound for the remaining cases is also less:
$(2 \cdot 2^{\frac{k+1}{2}} - 2) - (2(k-2h)2^h + 3 \cdot 2^h - 2) = (2 \cdot 2^{\frac{k+1}{2}} - 2) - 2(k-2h)2^h - 3 \cdot 2^h$
$= 2^h(2^{\frac{k+3}{2}-h} - 2k + 4h - 3) - 2.$
Letting $x = \frac{k+1}{2} - h$ transforms the above expression to $2^h(2^{x+1} - 4x - 1) - 2$.
It suffices to show that $2^{x+1} - 4x - 1 > 0$, which clearly holds for all $x \geq 3$. □

We are now ready to present the main result of this paper.

Theorem 8.1.12 *A pseudotree of maximum degree d and diameter at most k, where $d \geq 2$ and $k \geq 1$, has order at most:*

- $2k + 1$, *if* $d = 2$
- 3, *if* $k = 1$
- 5, *if* $k = 2$, $d = 3$
- 7, *if* $k = 3$, $d = 3$
- $\frac{d(d-1)^{\frac{k}{2}} - 2}{d-2}$, *if (*$k$ *is even,* $k \geq 2$, $d \geq 4$*)*
  or (k *is even,* $k \geq 4$, $d = 3$*)*

- $2 \cdot 2^{\frac{k+1}{2}} - 2$, *if* k *is odd,* $k \geq 5$, $d = 3$
- $3(d-1)^{\frac{k-1}{2}}$, *if* k *is odd,* $k \geq 3$, $d \geq 4$

Proof For $d = 2$ the only valid graph is a cycle and the extremal case is the C_{2k+1}.

For $k = 1$ the only valid graph is C_3.

For $k = 2$ and $d = 3$ it is easy to see that the extremal case is the C_5.

(by considering cases according to the girth of the construction)

For $k = 3$ and $d = 3$ it is easy to see that the extremal case is the C_7.

(by considering cases according to the girth of the construction)

The fifth case follows from Lemma 8.1.7.

The sixth case follows from Lemma 8.1.11.

The last case follows from Lemma 8.1.9. □

8.2 Maximizing the girth of a planar graph under girth constraints

We consider the EX-problem, a problem which asks given a graph of order n what is the maximum number of edges, denoted by $ex(n;t)$, that can exist in the graph such that it contains no cycle C_k, where $3 \leq k \leq t$. The set of those extremal C_t-free graphs (not to be confused with graphs that have no cycles of length t) is denoted by $EX(n;t)$. Erdős [105] initially posed the problem with $t = 4$. Since then a lot of research has been done trying to obtain exact solutions for the problem on general graphs, or obtaining good lower and upper bounds [1, 2, 3, 4, 15, 17, 46, 93, 127, 128, 176, 229, 230].

Research has also concentrated around special types of graphs, such as bipartite graphs [126, 16]. For $n \leq t$, it is easy to show that all trees including stars $K_{1,n-1}$ and paths P_n are extremal graphs and $ex(n;t) = n-1$. For $t+1 \leq n \leq \lfloor 3t/2 \rfloor$ the cycles C_n are extremal graphs and $ex(n;t) = n$. These values are included in the tables as folklore. The problem of finding the extremal number $ex(n;3)$ is solved by Mantel's theorem [190] which states that the maximum size of a triangle free graph of order n is $\lfloor n^2/4 \rfloor$. The extremal graphs $EX(n;3)$ are the complete bipartite graphs $K_{\lfloor n/2 \rfloor \lceil n/2 \rceil}$. For $t \geq 4$ no exact general formula is known for $ex(n;t)$. There are however general lower and upper bounds for $t = 4$. Currently the best lower bound is $\frac{n^{3/2}}{2\sqrt{2}}$, given by Garnick and Neuwejaar [128] and the best upper bound is $\frac{1}{2}n\sqrt{n-1}$, given by Garnick, Kwong and Lazebnik [127].

The bipartite case of the problem has been also studied. Early work had concentrated on avoiding cycles up to length 4 [37] or 6 [91, 92, 179, 199]. Later researchers tried to give bounds for general t. Hoory [147] and Lam [177] investigated the maximum number of edges in a

bipartite graph of arbitrary girth g whose left and right sides are of size n_L and n_R and gave some bounds. Later, Lam [178] improved his results and showed that if G has no cycles of length 2ℓ, for all $\ell \in [2, 2k]$, and $n_L \leq n_R$, then its size is less than $n_R^{\frac{1}{2}} + n_L^{\frac{k+1}{2k}} + O(n_L)$. Furthermore, Balbuena et al. [16, 126] showed that an extremal bipartite graph of girth at least $2t+2$ must contain a C_{2t+2} whenever $t = 2, 3$ or when m and n are large enough in comparison with t. They also gave the exact value of the extremal function $ex(m, n; \{C_4, \ldots, C_{2t}\})$ in the case that $m = n = 2t$ and in the case that $m = n = 2t + 1$ and showed that all extremal graphs are maximally connected. A similar problem asks for the maximum size of a graph of given order without a C_{2k}. Erdős [103] showed some bounds and conjectures for the case of $k = 3$; one of these conjectures has been proved to be true later by Györi [138]. Sárközy et al. [218] investigated a special case of this problem giving an upper bound. Finally, Naor et al. [198] gave an upper bound on the number of edges in a bipartite graph without a cycle of length $2k$.

A relevant problem is whether or not a given graph contains cycles of all lengths [30]. Such graphs are called pancyclic and much research has been concentrated around them as well. There are also obvious connections with the cage problem [110], also known as the degree/girth problem, as well as with the Zarankiewicz problem. Goddard et al. [130] investigated the case when the subgraph to be avoided is a C_4, thus showing some extremal values for the C_t-free problem in the case that $t = 4$.

In this section we consider the EX-problem for structures that have not been studied before, planar graphs and special types of planar graphs such as pseudotrees, cacti, Halin, generalized Halin graphs and graphs lying in an infinite square grid, giving the extremal numbers and some constructions [66]. We also consider the problem on bipartite graphs,

based on results and techniques from [66]. We give the extremal cases for a bipartite graph of low order and girth, where $1 \leq n \leq 20$ and $4 \leq t \leq 10$, except for a few instances of the problem. We use these results to obtain the extremal cases for bipartite graphs of high girth and for the problem in the general case.

8.2.1 Pseudotrees and Cacti

For a pseudotree the situation is easy to handle as the removal of one edge from the cycle of the pseudotree leaves a tree.

Theorem 8.2.1 *An extremal pseudotree of size n and girth at least $t+1$ has $ex(n,t) = n$, where $3 \leq t \leq n-1$.*

Proof The only limitation is that the cycle of the pseudotree must have length at least $t+1$. It is then clear that $|E| = n$ as removal of an edge from the cycle of the pseudotree leaves a tree which has $n-1$ edges, no matter the arrangement of the vertices lying on the hanging subtrees of the cycle. □

Cacti graphs are more complex than pseudotrees as they are allowed to have more than one cycle.

Theorem 8.2.2 *An extremal cactus graph of size n and girth at least $t+1$ has:*

$$ex(n,t) = \begin{cases} \lfloor \frac{n-t-1}{t} \rfloor + n, & 3 \leq t \leq n-1 \\ n-1, & t \geq n \end{cases}$$

Proof The number of edges of the cactus graph is:

$$|E| = (\text{number of cycles in the graph}) + n - 1,$$

as removal of an edge from each cycle of the cactus graph leaves a tree which has $n-1$ edges (or consider Euler's formula with n vertices and

number of faces equal to the number of cycles in the graph plus the unbounded face).

Maximizing E is then equivalent to maximizing the number of cycles in the cactus graph.

Cycles of the cactus graph must have length at least $t+1$.

Therefore we can have at most $\lfloor \frac{n-t-1}{t} \rfloor + 1$ cycles ($t+1$ vertices make the first cycle, then the remaining cycles are made by using t vertices).

So $ex(n,t) = \lfloor \frac{n-(t+1)}{t} \rfloor + n$. □

8.2.2 Generalized Halin graphs

Halin graphs are another tree-like structure. The limitation for the minimum degree of their vertices to be more than 2 forces the existence of a C_3. Actually it has been proved that Halin graphs are almost pancyclic, i.e. they contain cycles of all lengths $3 \leq l \leq n$ except possibly for one even value of l [31]. This fact implies the following theorem.

Theorem 8.2.3 *There are no extremal C_t-free Halin graphs for any $t \geq 3$.*

As the restriction for the minimum degree of a Halin graph to be more than 2 leaves no space for our problem we consider the generalized Halin graph, where we allow vertices of the graph to have degree 2. We relate the number of edges of the graph to the number of leaves of the ancestor tree of the Halin graph thus being able to derive conclusions.

Lemma 8.2.4 *A generalized Halin graph of order n and girth at least $t+1$ has at most $\lfloor \frac{2n-2}{t} \rfloor + n - 1$ edges if $3 \leq t \leq \lfloor \frac{2n-2}{t} \rfloor - 1$.*

Proof The number of edges of a generalized Halin graph is $|E|$ =length of cycle joining the leaves of the Halin graph + number of edges in the ancestor tree = number of leaves in the ancestor tree + $n-1$,

as the generalized Halin graph is made from an ancestor tree which has $n-1$ edges and one edge per leaf which join the tree leaves in a cycle. Maximizing E is then equivalent to maximizing the number of leaves in the ancestor tree.

The number of leaves of the generalized Halin graph is the number of its faces (excluding the outer one). Maximizing E is then equivalent to maximizing the number of its faces.

One can observe that every generalized Halin graph has all its nodes attached to the cycles mentioned above except if it is a path.

That path gives no contribution to the number of the faces formed, therefore we can consider only the rest of the cases.

Let us denote the boundaries (cycles) of the faces based on the ith and $i+1$th leaf as CY_i (leaves are ordered in the way they are found in a post order traversal of the ancestor tree of the Halin graph).

By counting the number of nodes of each such cycle we observe that each node is counted twice except maybe their roots.

The root is counted as many times as is the number of cycles hanging from it.

Inner roots are counted as many times as is the number of cycles hanging from them plus 2 (for their neighbour cycles).

Let the number of the cycles mentioned above be c. Then $\sum_{i=1}^{c} |CY_i| =$ 2(nodes that are counted twice) + number of roots$= 2(n-1) + c = 2n + c - 2$

Cycles of the generalized Halin graph must have length at least $t+1$.

Therefore $2n + c - 2 \geq c(t+1)$

and $c \leq \lfloor \frac{2n-2}{t} \rfloor$ □

Lemma 8.2.5 *A generalized Halin graph of size n and girth at least $t+1$ can have $\lfloor \frac{2n-2}{t} \rfloor + n - 1$ edges if $3 \leq t \leq \lfloor \frac{2n-2}{t} \rfloor - 1$.*

Proof It is easy to see that the bound introduced in the theorem above

is achieved by the following construction:

Consider the root connected to paths of length $\lceil \frac{t}{2} \rceil$, $\lfloor \frac{t}{2} \rfloor$, $\lceil \frac{t}{2} \rceil$, $\lfloor \frac{t}{2} \rfloor$,...

Any remaining vertices are added to the last path (see also Figure 8.4).

Then $c = 2\lfloor \frac{n-1}{t} \rfloor + \lceil (n-1) \mod t - \lceil \frac{t}{2} \rceil + 1) \rceil = \lfloor \frac{2n-2}{t} \rfloor$

(When $(n-1) \mod t \geq \lceil \frac{t}{2} \rceil$ the second term of the expression is 1, 0 otherwise)

So $ex(n,t) = \lfloor \frac{2n-2}{t} \rfloor + n - 1$.

□

Theorem 8.2.6 *A generalized Halin graph of size n and girth at least $t+1$ has*

$$ex(n,t) = \begin{cases} \lfloor \frac{2n-2}{t} \rfloor + n - 1, & 3 \leq t \leq \lfloor \frac{2n-2}{t} \rfloor - 1 \\ n-1, & t \geq \lfloor \frac{2n-2}{t} \rfloor \end{cases}$$

Proof By Lemmas 8.2.4 and 8.2.5 we get the above result for $3 \leq t \leq \lfloor \frac{2n-2}{t} \rfloor - 1$. However if $t \geq \lfloor \frac{2n-2}{t} \rfloor$ the only valid construction is P_n, a path starting from the root ending at a single leaf, as it has no connected leaves which would result to a cycle of length at least $\lfloor \frac{2n-2}{t} \rfloor$. □

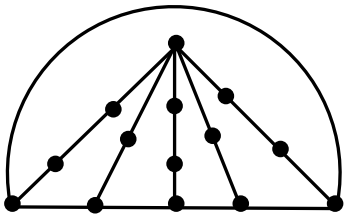


Figure 8.4: An extremal generalized Halin graph of order 14 and girth at least 5

8.2.3 Rectangular grid graphs

It appears that to maximize the number of edges of a graph of order n in the grid we need to minimize the perimeter of a closed shape containing

all n vertices. Thus the situation becomes easier to handle and we are able to draw conclusions for extremal numbers and give some constructions. We note that we consider only cases for odd t, as due to the fact that no odd cycles can exist in the grid $ex(n, 2k) = ex(n, 2k+1)$ for $k \geq 2$.

Theorem 8.2.7 *A graph of order n and girth at least $t+1$ that lies in an infinite square grid can have at most:*

- $n-1$ *edges ,if* $1 \leq n \leq t$
- $\lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1$ *edges, if* $t+1 \leq n \leq \lceil \frac{t+1}{4} \rceil \lfloor \frac{t+1}{4} \rfloor$
- $\lfloor \frac{2n+2-2\lceil \sqrt{n} \rceil - 2\lfloor \sqrt{n} \rfloor}{t-1} \rfloor + n - 1$ *edges, if* $\lceil \frac{t+1}{4} \rceil \lfloor \frac{t+1}{4} \rfloor + 1 \leq n \leq \lceil \sqrt{n} \rceil \lfloor \sqrt{n} \rfloor$
- $\lfloor \frac{2n+2-4\lceil \sqrt{n} \rceil}{t-1} \rfloor + n - 1$ *edges, if* $\lceil \frac{t+1}{4} \rceil \lfloor \frac{t+1}{4} \rfloor + 1 \leq n$ *and* $\lceil \sqrt{n} \rceil \lfloor \sqrt{n} \rfloor + 1 \leq n \leq \lceil \sqrt{n} \rceil \lceil \sqrt{n} \rceil$

Proof For $1 \leq n \leq t$ our graph must be acyclic and any maximal acyclic graph is by definition a tree with $n-1$ edges. If $t+1 \leq n$ there must be cycles present in our graph.

Let us denote the number of faces of the graph by c.

Using Euler's formula for planar graphs we get $|E| = c + n - 1$.

By summing the edges of each of the boundaries of each face (except the outer face) we get:

$\sum_{i=1}^{c} |Cy_i| = 2x + y$, where x is the number of edges of the graph that are counted twice, y is the number of edges of the graph that are counted once and $\{Cy_1, Cy_2, \ldots, Cy_c\}$ is the set of cycles that are the boundaries the faces (except the outer face) present in our graph.

Let us denote the number of the edges that do not belong to any of $\{Cy_1, Cy_2, \ldots, Cy_c\}$ by h. As $|E| = x+y+h$, $\sum_{i=1}^{c} |Cy_i| = 2|E| - y - 2h$.

By Euler's formula we get: $\sum_{i=1}^{c} |Cy_i| = 2c + 2n - 2 - y - 2h$

As these cycles must have length at least $t+1$:

$c(t+1) \leq \sum_{i=1}^{c} |Cy_i| = 2c + 2n - 2 - y - 2h$

i.e. $c \leq \lfloor \frac{2n-2-y-2h}{t-1} \rfloor$

In order to maximize $|E|$ our graph must be connected (otherwise we can free a node and create more edges by connecting the rightmost node (take the lowest if they are many) of a maximally connected subgraph of G to the leftmost node (take the highest if they are many) of another maximally connected subgraph of G).

Then y is the perimeter of a two dimensional shape in the grid (excluding any hanging edges).

Hanging edges do not contribute to the number of faces formed and so we should avoid them (an even number of nodes can be added in an existing cycle with a side on the outer boundary, reducing the graph to another with one or none hanging edges). Therefore, in deriving an upper bound for E we assume that there exist at most one hanging vertex in our construction. It is easy to see that any such shape is contained in a rectangle which touches its leftmost, rightmost, top and bottom edges. This rectangle has same or smaller perimeter than the contained shape and maybe some more area.

It is crucial that the shape contains at least n vertices.

In the case that $h = 0$, minimizing y means that we choose a rectangle with sides $\lceil \sqrt{n} \rceil - 1$, $\lfloor \sqrt{n} \rfloor - 1$ and if we can not fit all n vertices in it we choose a square with sides of length $\lceil \sqrt{n} \rceil - 1$. In the case that $h = 1$, we can get no better upper bounds as we can maybe reduce y by 2 but we have to subtract 2 from the numerator of the expression for the upper bound. □

The following theorem suggests that the above upper bounds can be achieved if we somehow avoid no non-simple cycles of length less than $t + 1$ in our constructions.

Theorem 8.2.8 *The upper bounds of Theorem 8.2.7 can be achieved*

if no non-simple cycles (cycles other than the boundaries of the faces formed) of length less than $t+1$ are present in our constructions.

Proof To achieve the upper bounds of Theorem 8.2.7 we need to tessellate the rectangle mentioned in Theorem 8.2.7 with as many cycles of length $t+1$ as possible. That is considering as many cycles of area $\frac{t-1}{2}$ (i.e. enclosing exactly $\frac{t-1}{2}$ squares of the grid). Then we get $c \leq \lfloor \frac{\text{Area of rectangle}}{(t-1)/2} \rfloor$ and hence the above upper bounds. (as long as no non-simple cycles of length less than $t+1$ are present in our constructions) □

Below we show that the upper bounds that we have introduced can be achieved in certain cases.

Theorem 8.2.9 *The above upper bound can be achieved for $n \geq (\frac{t+1}{2})^2$. Therefore for $n \geq (\frac{t+1}{2})^2$:*

$$ex(n;t) = \begin{cases} \lfloor \frac{2n+2-2\lceil\sqrt{n}\rceil-2\lfloor\sqrt{n}\rfloor}{t-1} \rfloor + n - 1, \ n \leq \lceil\sqrt{n}\rceil\lfloor\sqrt{n}\rfloor \\ \lfloor \frac{2n+2-4\lceil\sqrt{n}\rceil}{t-1} \rfloor + n - 1, \ \lceil\sqrt{n}\rceil\lfloor\sqrt{n}\rfloor + 1 \leq n \leq \lceil\sqrt{n}\rceil\lceil\sqrt{n}\rceil \end{cases}$$

Proof To achieve the upper bound introduced in the above theorem we need to tessellate the rectangle mentioned above with as many cycles of length $t+1$ as possible as Theorem 8.2.8 suggests. This can be obtained by tiling the central $(\frac{t+1}{2}-1)$x$(\frac{t+1}{2}-1)$ square with $\frac{t+1}{2}-1$ cycles in the form of a column of thickness 1. We then spiral the rest of the cycles around it until all vertices are exhausted as shown in Figure 8.5. It is easy to see that no cycle of length less than $\frac{t+1}{2}$ is formed due to cycles touching each other. □

8.2.4 Planar graphs

Using similar arguments we are now able to give the extremal numbers and some extremal graphs for the problem on any planar graph.

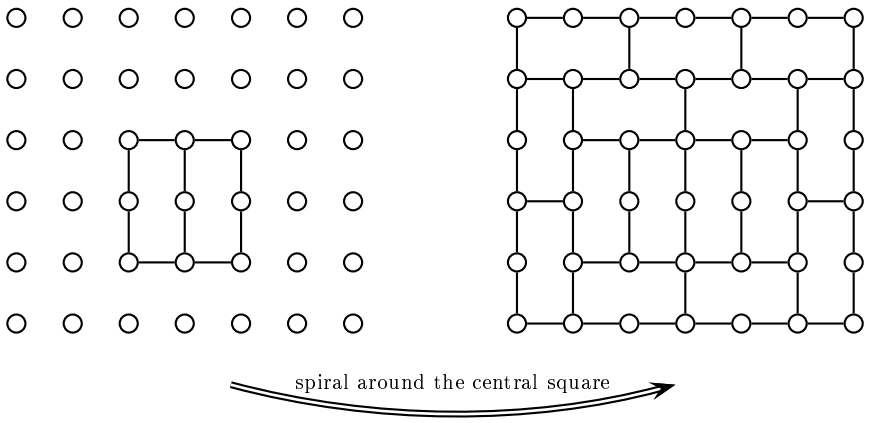


Figure 8.5: Construction of an extremal graph for $n = 42$ and girth at least 6

Lemma 8.2.10 *A planar graph of order n and girth at least $t+1$ can have at most:*

- $n-1$ *edges, if* $1 \leq n \leq t$
- $\lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1$ *edges, if* $t+1 \leq n$

Proof For $1 \leq n \leq t$ our graph must be acyclic and any maximal acyclic graph is by definition a tree with $n-1$ edges. If $t+1 \leq n$ there must be cycles present in our graph.

Let us denote the number of faces of the graph by c (excluding the outer face).

Using Euler's formula for planar graphs we get $|E| = c + n - 1$.

By summing the edges of each cycle (which is a boundary of a face) we get:

$\sum_{i=1}^{c} |Cy_i| = 2x + y$, where x is the number of edges of the graph that are counted twice and y the number of edges of the graph that are counted once.

Edges that do not belong to these cycles do not contribute to the number of faces formed in the graph and so we should not include them in our constructions. Any such hanging edge could be added in an exist-

ing cycle, thus reducing the graph to one with no hanging edges. Then $|E| = x + y$, $\sum_{i=1}^{c} |Cy_i| = 2|E| - y$.

By Euler's formula we get: $\sum_{i=1}^{c} |Cy_i| = 2c + 2n - 2 - y$

As cycles must have length at least $t + 1$:

$c(t + 1) \leq \sum_{i=1}^{c} |Cy_i| = 2c + 2n - 2 - y$

i.e. $c \leq \lfloor \frac{2n-2-y}{t-1} \rfloor$

Using similar arguments as in Theorem 8.2.7, our graph must be connected in order to maximize $|E|$. Then y is the perimeter of a two dimensional shape.

Following the restriction for the girth we get $y \geq t + 1$ and hence the above upper bound. □

Theorem 8.2.11 *The above upper bound can be achieved for any $n \geq 1$. Therefore:*

$$ex(n;t) = \begin{cases} n-1, & 1 \leq n \leq t \\ \lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1, & n \geq t+1 \end{cases}$$

Proof For $1 \leq n \leq t$ our graph must be acyclic and any maximal acyclic is by definition a tree with $n - 1$ edges, so any tree on n vertices gives a lower bound construction.

For $n \geq t + 1$ we consider the following cases:

- If t is odd the following construction gives the upper bound. Connect the first $t + 1$ vertices to make a C_{t+1}. Number those vertices from 1 to $t + 1$. Keep connecting paths of length $\frac{t-1}{2}$ on vertices 1 and $2 + \frac{t-1}{2}$. Insert any remaining vertices in the latest added path, obtaining $1 + \lfloor \frac{n-(t+1)}{\frac{t-1}{2}} \rfloor = \lfloor \frac{2n-t-3}{t-1} \rfloor$ cycles (see also Figure 8.6).

- If t is even the following construction gives the upper bound (actually it works also for odd t but the above construction is much simpler). Connect the first $t + 1$ vertices to make a C_{t+1}. Number

those vertices from 1 to $t+1$. Connect a path of length $\frac{t}{2}$ on vertices 1 and $1+\frac{t}{2}$ meeting the outer face. Connect a path of length $\frac{t}{2}-1$ starting from the second vertex of the latest added path (count from the start of the path not from the vertex $1+\frac{t}{2}$ or $2+\frac{t}{2}$ which is its end) ending at vertex $2+\frac{t}{2}$ if latest added path was ending at vertex $1+\frac{t}{2}$ or otherwise at vertex $1+\frac{t}{2}$, meeting the outer face. Connect a path of length $\frac{t}{2}$ on the same vertices as the previous path, meeting the outer face. Repeat until no vertices are left to make such paths. Insert any remaining vertices in the latest added path, obtaining $2\lfloor\frac{n-\frac{t-1}{2}-2}{t-1}\rfloor + I(n-\frac{t-1}{2}-2 \bmod t-1 \geq \frac{t}{2}-1)$ cycles.

The number of cycles can be at most $\lfloor\frac{2n-t-3}{t-1}\rfloor = \lfloor\frac{2n-t-4}{t-1}\rfloor$
$= 2\lfloor\frac{n-\frac{t}{2}-2}{t-1}\rfloor + I(\frac{n-\frac{t-1}{2}-2}{t-1} \geq \frac{1}{2})$
$= 2\lfloor\frac{n-\frac{t}{2}-2}{t-1}\rfloor + I(n-\frac{t-1}{2}-2 \bmod t-1 \geq \frac{t}{2}-1)$
(see also Figure 8.7).

It is easy to see that no cycle of length less than $\frac{t+1}{2}$ is formed due to cycles touching each other. □

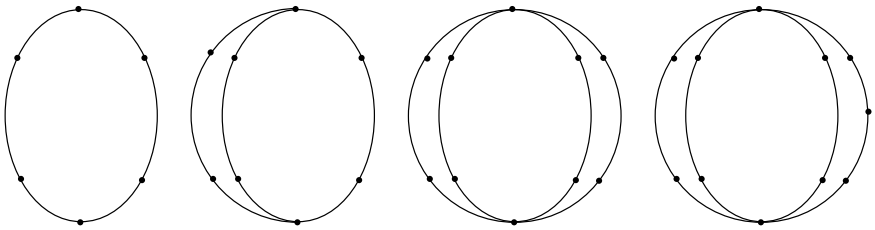


Figure 8.6: Step by step construction of an extremal planar graph of order 11 and girth at least 6

By considering planar graphs we are now able to give a lower bound for the problem on its general case.

Theorem 8.2.12 *A graph of order n and girth at least $t+1$ has $ex(n,t)$ at least:*

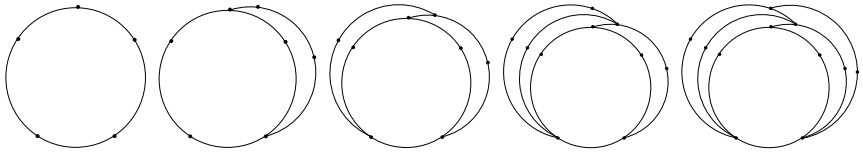


Figure 8.7: Step by step construction of an extremal planar graph of order 11 and girth at least 5

- $n-1$, *if* $1 \leq n \leq t$
- $\lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1$, *if* $t+1 \leq n$

Proof Consider the planar graph constructions used in the proof of Theorem 8.2.11. □

8.3 C_t-free bipartite graphs

8.3.1 C_t-free bipartite graphs of low order and girth

Before proceeding with our results we need to recall the following theorems.

The following theorem gives the necessary and sufficient condition for a graph to be planar.

Theorem 8.3.1 ***(Kuratowski's theorem)*** *A finite graph G is planar if and only if it contains no subgraph that is a subdivided K_5 or a subdivided $K_{3,3}$.*

The following theorem gives us the $ex(n;t)$ numbers in the cases that our graph is planar.

Theorem 8.3.2 *[66] When G is planar we have:*

$$ex(n;t) = \begin{cases} n-1, & 1 \leq n \leq t \\ \lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1, & n \geq t+1 \end{cases}$$

The extremal construction for planar graphs can be extended for a bipartite planar graph, thus helping us to get a lower bound for the EX problem on bipartite graphs.

Corollary 8.3.3 *[66] When t is odd, $t \geq 5$ and G is both bipartite and planar we have $ex(n;t) = ex(n;t-1)$. Furthermore when t is odd, $t \geq 3$ and G is both bipartite and planar, we have*

$$ex(n;t) = \begin{cases} n-1, & 1 \leq n \leq t \\ \lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1, & n \geq t+1 \end{cases}$$

Proof As in Theorem 8.2.11, for $1 \leq n \leq t$, our graph must be any tree on n vertices and it is well known that a tree is both planar and bipartite. For $n \geq t+1$ Theorem 8.2.11 provides us with an extremal graph constructed by using the following steps.

- Connect the first $t+1$ vertices to make a C_{t+1}.
- Number those vertices from 1 to $t+1$.
- Keep connecting paths of length $\frac{t-1}{2}$ on vertices 1 and $2 + \frac{t-1}{2}$.
- Insert any remaining vertices in the latest added path, obtaining $\lfloor \frac{n-\frac{t-1}{2}-2}{\frac{t-1}{2}} \rfloor = \lfloor \frac{2n-t-3}{t-1} \rfloor$ cycles.

In this case we skip the last step and instead we connect any remaining vertices in a path starting from vertex 1 (see also Figure 8.8.) □

In the case that the order of our graphs is small the search for extremal graphs can be limited to planar graphs with the help of Kuratowski's theorem, as shown below. Therefore Theorem 8.2.11 gives us the extremal number when n is small enough.

The following lemmas investigate the appearance of Kuratowski subgraphs, subgraphs that are subdivisions of a $K_{3,3}$ or a K_5, in bipartite graphs of girth at least 8.

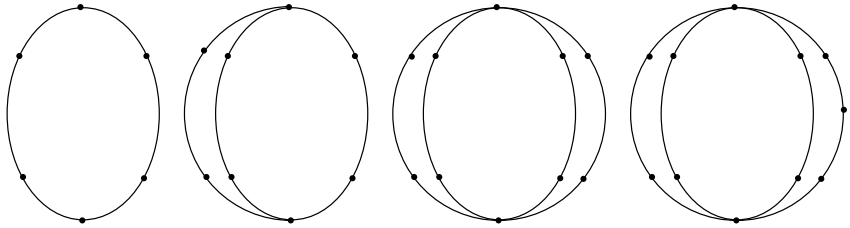


Figure 8.8: Step by step construction of an extremal bipartite planar graph of order 11 and girth at least 6, where a two colouring of the vertices of the graph is shown in the last step.

Lemma 8.3.4 *There is no subdivided $K_{3,3}$ in a bipartite graph of girth at least 8 and order less than 15.*

Proof There exist $\binom{3}{2} \cdot \binom{3}{2} = 9$ C_4s in $K_{3,3}$. By subdividing a $K_{3,3}$ such that it has girth at least 8, those C_4s result in cycles of length at least 8. Added vertices are part of 4 such cycles. Therefore $n \geq 6 + \frac{9 \cdot 4}{4} = 15$. □

Lemma 8.3.5 *There is no subdivided K_5 in a graph of girth at least 8 and order less than 22.*

Proof There exist $\binom{5}{3} = 10$ triangles in K_5. By subdividing a K_5 such that it has girth at least 8, those triangles result in cycles of length at least 8. Added vertices are part of 3 cycles. Therefore $n \geq 5 + \frac{5 \cdot 10}{3} = 21\frac{2}{3}$. □

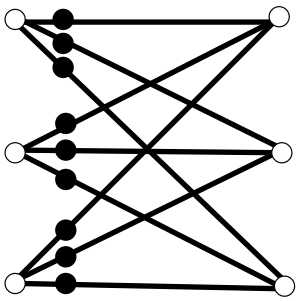


Figure 8.9: An extremal bipartite graph of order 15 and girth at least 8 where a two colouring of the vertices of the graph is shown.

The following lemma investigates the appearance of Kuratowski subgraphs in bipartite graphs of girth at least 10.

Lemma 8.3.6 *There is no subdivided $K_{3,3}$ in a bipartite graph of girth at least* 10 *and order less than* 21.

Proof There exist $\binom{3}{2} \cdot \binom{3}{2} = 9$ C_4s in $K_{3,3}$. By subdividing a $K_{3,3}$ such that it has girth at least 10, those C_4s result in cycles of length at least 10. Added vertices are part of 4 such cycles. Therefore $n \geq 6 + \frac{9 \cdot 6}{4} = 19\frac{1}{2}$.

In the case that $n = 20$ we will try to see if such a graph is feasible. Suppose we have a $K_{3,3}$ with its vertices partitioned in two sets $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$. Let $x_{i,j}$ = number of vertices that subdivide the edge $a_i b_j$, where $i, j \in \{1, 2, 3\}$. Suppose that at least one edge is not subdivided, i.e. without loss of generality we assume that $x_{1,1} = 0$. By considering the C_4s created by a_1, a_2, b_1, b_2 and a_1, a_3, b_1, b_3 and having in mind that each C_4 must be subdivided by at least 6 vertices we get that $x_{2,3} + x_{3,2} = 2$. By considering the C_4s created by a_1, a_3, b_1, b_2 and a_1, a_2, b_1, b_3 and the subdivision restrictions we get that $x_{1,2} + x_{3,1} + x_{3,2} + x_{1,3} + x_{2,1} + x_{2,3} \geq 10$. Combining this with the restrictions for the C_4 created by a_1, a_2, b_1, b_2 we get that $x_{1,2} + x_{3,1} + x_{3,2} + x_{1,3} + x_{2,1} + x_{2,3} + x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2} \geq 16$ (contradiction as then $n \geq 22$). □

With the help of the above lemmas we are now in a position to show the main result of this section.

Theorem 8.3.7 *The $ex(n; t)$ values for a bipartite graph of order $1 \leq n \leq 20$ and girth at least $t + 1$, where $4 \leq t \leq 10$ except for $n \in \{17, 18, 19, 20\}$ and $t \in \{6, 7\}$, are given in Table 8.1. In the case that $n \in \{17, 18, 19, 20\}$ and $t \in \{6, 7\}$ we give a lower bound attained by the planar construction of Corollary 8.3.3.*

Proof is a consequence of Lemmas 3.4-3.6. In the case that $t = 4$ the extremal values are the Zarankiwickz numbers $z(n; K_{2,2}) = z(n; C_4)$

[130]. In the case that $n = 15$ and $t = 6$, the non planar construction of Figure 8.9 gives the extremal value. In the case that $n = 16$ and $t = 6$ the planar construction of Corollary 8.3.3 gives $ex(16; 6) \geq 19$. A subdivided $K_{3,3}$ with 16 nodes has 19 edges as well. The remaining case is a subdivided $K_{3,3}$ of order 15 with a node x attached on it. If the degree of x is 1 then $m = 19$. Let $d(x) \geq 2$ and suppose that we have a $K_{3,3}$ with its vertices partitioned into two sets $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$. Then x must be attached to at least one edge of the subdivided $K_{3,3}$. Suppose it is attached to at least 2 edges of the subdivided $K_{3,3}$. W.l.o.g. we consider x to be attached to a_3b_3 and a_2b_3. The subdivided cycles $a_2b_3a_3x$ and $a_1b_1a_2b_2$ must have order at least 8 which leaves the subdivided path $b_1a_3b_2$ to have order 3 or 4. In the case that the subdivided path $b_1a_3b_2$ has order 3, the subdivided paths $b_1a_1b_2$ and $b_1a_2b_2$ must each have order at least 7, which gives a contradiction. In the case that the subdivided path $b_1a_3b_2$ has order 4 the subdivided paths $b_1a_1b_2$ and $b_1a_2b_2$ must each have order at least 6, which gives a contradiction. The proof is similar when x is attached to only one edge of the subdivided $K_{3,3}$. □

8.3.2 C_t-free bipartite graphs of high girth

Using similar techniques we are now able to give the extremal values when t is large compared to n.

The following lemmas investigate the appearance of Kuratowski subgraphs in bipartite graphs of girth at least $t + 1$.

Lemma 8.3.8 *There is no subdivided $K_{3,3}$ in a bipartite graph of girth at least $t + 1$ and order less than $\frac{9t-21}{4}$.*

Proof There exist $\binom{3}{2} \cdot \binom{3}{2} = 9$ C_4s in $K_{3,3}$. By subdividing a $K_{3,3}$ such that it has girth at least $t+1$, those C_4s result in cycles of length at least

n/t	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1
3	2	2	2	2	2	2	2
4	3	3	3	3	3	3	3
5	4	4	4	4	4	4	4
6	6	6	5	5	5	5	5
7	7	7	6	6	6	6	6
8	9	9	8	8	7	7	7
9	10	10	9	9	8	8	8
10	12	12	10	10	10	10	9
11	14	14	12	12	11	11	10
12	16	16	13	13	12	12	12
13	18	18	14	14	13	13	13
14	21	21	16	16	15	15	14
15	22	22	18	18	16	16	15
16	24	24	19	19	17	17	16
17	26	26	20*	20*	18	18	18
18	29	29	21*	21*	20	20	19
19	31	31	22*	22*	21	21	20
20	34	34	24*	24*	22	22	21

Table 8.1: Summary of exact values of $ex(n;t)$ for bipartite graphs. Starred values indicate a lower bound.

$t+1$. Added vertices are part of 4 such cycles. Therefore $n \geq 6+\frac{9\cdot(t-5)}{4} = \frac{9t-21}{4}$. □

Lemma 8.3.9 *There is no subdivided K_5 in a graph of girth at least $t+1$ and order less than $\frac{10t-5}{3}$.*

Proof There exist $\binom{5}{3} = 10$ triangles in K_5. By subdividing a K_5 such that it has girth at least $t+1$, those triangles result in cycles of length at least $t+1$. Added vertices are part of 3 cycles. Therefore $n \geq 5+\frac{10\cdot(t-2)}{3} = \frac{10t-5}{3}$. □

Based on the above lemmas we are now able to give the extremal values for bipartite graphs of high girth. It is easy to see that these are the extremal values for the problem on general graphs as well.

Theorem 8.3.10 *Whenever* $t > \frac{4n+21}{9}$, $ex(n;t)$ *is given by the construction of Corollary 8.3.3, that is,*

$$ex(n;t) = \begin{cases} n-1, & 1 \leq n \leq t \\ \lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1, & n \geq t+1 \end{cases}$$

Proof As deduced by the above lemmas, whenever $n < \frac{10t-5}{3}$ (and $n < \frac{9t-21}{4}$) then $ex(n;t)$ is given by the construction of Corollary 8.3.3.

It is enough to consider the second inequality only since when $t > 0$ then we have

$$\frac{10t-5}{3} \geq \frac{9t-21}{4}$$

□

8.3.3 Conclusion

In this section we have considered the EX problem for bipartite graphs. We have given the extremal cases for a bipartite graph of low order and girth, i.e., when $1 \leq n \leq 20$ and $4 \leq t \leq 10$ (Table 8.1) except for $n \in \{17, 18, 19, 20\}$ and $t \in \{6, 7\}$. We have used our results to obtain the extremal cases for bipartite graphs of high girth and for the problem in the general case. More specifically, whenever $t > \frac{4n+21}{9}$, we found $ex(n;t)$, as given by the construction of Corollary 8.3.3, i.e. ,

$$ex(n;t) = \begin{cases} n-1, & 1 \leq n \leq t \\ \lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1, & n \geq t+1 \end{cases}$$

When G is bipartite, $t > \frac{4n+21}{9}$, t is even and $t \geq 6$ then $ex(n;t)$ is given by the above construction; in the case that t is odd and $t \geq 5$ we have $ex(n;t) = ex(n;t-1)$.

8.4 Bipartite Ramsey numbers involving stars, stripes and trees

Ramsey theory explores the question of how big a structure must be to contain a certain substructure or substructures (for a list of applications see [212]).

Ramsey [206] showed that in a blue-red colouring of the edges of a sufficiently large complete graph there must exist either a blue or a red complete subgraph of a given order. The minimum order of a complete graph that must achieve that is known as a Ramsey number. Since then large amount of research has been done trying to obtain exact values for Ramsey numbers, or to obtain good lower and upper bounds[205].

There are many generalizations of Ramsey theory. Multicolour Ramsey theory deals with the same problem involving more than two colours. Infinite Ramsey theory investigates similar problems on infinite graphs. Ramsey numbers also exist for monochromatic graphs other than complete subgraphs, e.g. trees, stars, bipartite graphs, cycles, paths, etc. Bipartite Ramsey problems deal with the same questions but the graph explored is the complete bipartite graph instead of the complete graph. Additionally, there are many similar questions for directed graphs.

The bipartite case has been studied extensively. In particular, research has been done to obtain exact values for small Ramsey numbers ([22, 109, 141, 183]). A first general upper bound was given by Irving [156] by exploring the similarity of the problem with Zarankiewicz‘s problem. Subsequent work on general bounds for the problem was given by

Thomason et al. [231], by Hattingh et al. [141], by Goddard et al. [130], by Caro et al. [44], by Conlon [73] and Lin et al. [182]. Exact solutions were given for simpler cases of the problem such as path-path bipartite Ramsey numbers [111, 137], star-star bipartite Ramsey numbers [183], star-path bipartite Ramsey numbers [142], $K_{2,2}$-$K_{1,n}$ and $K_{2,2}$-$K_{2,n}$ bipartite Ramsey numbers [42], C_{2m}-$K_{2,2}$ bipartite Ramsey numbers [213] and bipartite Ramsey numbers for multiple copies of $K_{2,2}$ [143]. Some variations of the bipartite case such as multicolour problems [43, 70] and rainbow colouring problems [108] (the bipartite rainbow ramsey number $BRR(G_1; G_2)$ is the smallest integer n such that any colouring of the edges of K_{nn} with any number of colours contains a monochromatic copy of G_1 or a rainbow copy of G_2, where rainbow means that no two edges of the graph can have the same colour) have been also studied.

In this section we consider special cases of the bipartite Ramsey problem that have not been studied before. More specifically we investigate the appearance of simpler mono-chromatic graphs such as stripes, stars and trees under a 2-colouring of the edges of a bipartite graph. We give the Ramsey numbers $R_b(mP_2, nP_2)$, $R_b(T_m, T_n)$, $R_b(S_m, nP_2)$, $R_b(T_m, nP_2)$ and $R_b(S_m, T_n)$ [67].

8.4.1 Bipartite Ramsey numbers involving stars, stripes and trees

In this subsection we will give solutions to the problems that we are considering. For the first four problems we first give an upper bound and then we prove that it is tight. However the bipartite Ramsey numbers for trees appear to be smaller in the case that both of the considered trees are of even order. In the last case we first give an upper bound, then we show when it can be achieved and we give the exact solutions for the rest of its cases.

The solution to the bipartite Ramsey stripe problem is an immediate consequence of the following theorem.

Theorem 8.4.1 $R_b(mP_2, nP_2) = m + n - 1$.

Proof We will first prove that $R_b(mP_2, nP_2) \leq m+n-1$ by considering a 2-colouring of $K_{b,b}$, where $b = m + n - 1$. We pick a maximal $K_{k,k}$ containing a blue kP_2. If $k \geq m$ we have a blue mP_2 otherwise maximality forces a red $K_{b-k,b-k}$ made from the remaining vertices. That means a red $(b-k)P_2$. But $k < m$ and so $b - k > (n + m - 1) - m$. Hence $b - k \geq n$.

The following lower bound shows that $R_b(mP_2, nP_2) > m + n - 2$. We consider the following 2-colouring of $K_{m+n-2,m+n-2}$ (see also Figure 8.10):

- Let the independent sets of $K_{m+n-2,m+n-2}$ be A and B.
- We colour the edges joining the first $m - 1$ vertices of A to the vertices of B.
- We colour the rest of the edges red.

□

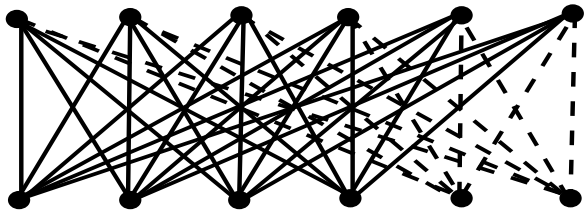


Figure 8.10: A 2 colouring of $K_{6,6}$ without a blue (continuous line) $5P_2$ or a red (dashed line) $3P_2$

The solution for the bipartite Ramsey tree problem is broken into five lemmas as shown below, depending on whether the considered trees are both of even order and whether the orders of the considered trees are close enough.

Theorem 8.4.2

$$R_b(T_m, T_n) = \begin{cases} m-1, & m = n = 2k, \\ & k \in \mathbb{Z}^+ \\ \max(\min(m,n), \max(\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil)), & \text{otherwise.} \end{cases}$$

Proof W.l.o.g. we consider $2 \leq m \leq n$. The general upper bound is given by Lemma 8.4.3. The construction of Lemma 8.4.4 gives the lower bound for the case $m < \lceil \frac{n}{2} \rceil$. The construction of Lemma 8.4.5 gives the lower bound for the case $m \geq \lceil \frac{n}{2} \rceil$, with the restriction that m and n cannot be equal if they are both even. A stricter upper bound in the case that m is even and $m = n$ is given by Lemma 8.4.6 and finally the construction of Lemma 8.4.7 gives the lower bound for this case. □

The following lemma gives us the general upper bound for this problem.

Lemma 8.4.3 *$R_b(T_m, T_n) \leq max(m, \lceil \frac{n}{2} \rceil)$, where $2 \leq m \leq n$.*

Proof Consider a 2-colouring of $K_{b,b}$, where $b = max(m, \lceil \frac{n}{2} \rceil)$. Let the independent sets of $K_{b,b}$ be A and B. Consider a maximal red T_k. W.l.o.g. say it is made by the first x vertices in A and the first $k - x$ vertices of B. If $k = b$ we get a red T_{2b} (i.e. at least a T_n), otherwise maximality forces a blue $K_{b-x,k-x}$ (composed by the last $b - x$ vertices of A and the first $k - x$ vertices of B) and a blue $K_{x,b+x-k}$ (composed by the first x vertices of A and the last $b + x - k$ vertices of B) and so a blue T_{b-2x+k} and a blue T_{b+2x-k}.
Therefore, there is at least one blue T_b (even k) or at least one blue T_{b+1} (odd k) and hence in either case there exists at least a blue T_m. □

The following lemma gives us a lower bound for this problem for the case where $2 \leq m < \lceil \frac{n}{2} \rceil$.

Lemma 8.4.4 *$R_b(T_m, T_n) > \lceil \frac{n}{2} \rceil - 1$, where $2 \leq m < \lceil \frac{n}{2} \rceil$.*

Proof A red $K_{\lceil \frac{n}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1}$ contains at most a red T_{n-2} (even n) or at most a red T_{n-1}(odd n). □

The following lemma gives us a lower bound for this problem for the case where $2 \leq \lceil \frac{n}{2} \rceil \leq m < n$ or $2 < n = m = 2k + 1$ with $k \in \mathbb{Z}^+$.

Lemma 8.4.5 *$R_b(T_m, T_n) > m - 1$, where $2 \leq \lceil \frac{n}{2} \rceil \leq m < n$ or $2 < n = m = 2k + 1$ with $k \in \mathbb{Z}^+$.*

Proof Consider the following 2-colouring of $K_{m-1,m-1}$ (see also Figure 8.11a):

- Let the independent sets of $K_{m-1,m-1}$ be A and B.
- Colour the edges joining the first $\lfloor \frac{m-1}{2} \rfloor$ vertices of A with the first $\lceil \frac{m-1}{2} \rceil$ vertices of B blue.
- Colour the edges joining the last $\lceil \frac{m-1}{2} \rceil$ vertices of A with the last $\lfloor \frac{m-1}{2} \rfloor$ vertices of B blue.
- Colour the rest of the edges red.

□

The following lemma gives us a lower upper bound for this problem for the case where $m = n = 2k$ with $k \in \mathbb{Z}^+$.

Lemma 8.4.6 *$R_b(T_m, T_m) \leq m - 1$ if m is even.*

Proof Consider a 2-colouring of $K_{m-1,m-1}$. Let the independent sets of $K_{m-1,m-1}$ be A and B. By previous lemma it contains either a blue $T_m - 1$ or a red $T_m - 1$. W.l.o.g. say a blue $T_m - 1$ made by the first x vertices in A and the first $m - 1 - x$ vertices of B. Maximality forces a red $K_{m-1-x,m-1-x}$ (composed by the last $m - 1 - x$ vertices of A and

the first $m-1-x$ vertices of B) and a red $K_{x,x}$ (composed by the first x vertices of A and the last x vertices of B) and so a red $T_{2(m-1-x)}$ and a red T_{2x}. Therefore, there is at least one red $T_{2\lceil\frac{m-1}{2}\rceil} = T_m$ (as $m-1$ is odd). □

The following lemma shows that the upper bound established in the previous lemma can be achieved.

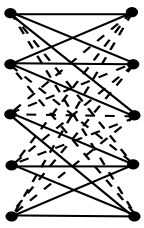
Lemma 8.4.7 *$R_b(T_m, T_m) > m-2$ if $n = m = 2k$.*

Proof Let $b = min(m, n)$.

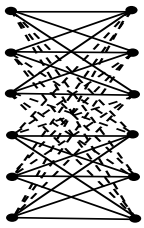
Consider the following 2-colouring of $K_{m-2,m-2}$ (see also Figure 8.11b):

- Let the independent sets of $K_{m-2,m-2}$ be A and B.
- Colour the edges joining the first $\lfloor\frac{m-1}{2}\rfloor$ vertices of A with the first $\lfloor\frac{m-1}{2}\rfloor$ vertices of B blue.
- Colour the edges joining the last $\lfloor\frac{m-1}{2}\rfloor$ vertices of A with the last $\lfloor\frac{m-1}{2}\rfloor$ vertices of B blue.
- Colour the rest of the edges red.

□



(a) A 2 colouring of $K_{5,5}$ without a blue (continuous line) T_6 or a red (dashed line) T_8



(b) A 2 colouring of $K_{6,6}$ without a blue (continuous line) T_8 or a red (dashed line) $T8$

Figure 8.11: Lower bound constructions considered in Theorem 8.4.2

The solution to the bipartite star vs stripes problem is an immediate consequence of the following theorem.

Theorem 8.4.8 $R_b(S_m, nP_2) = m + \lfloor \frac{n-1}{2} \rfloor - 1$.

Proof We will first prove that $R_b(S_m, nP_2) \leq m + \lfloor \frac{n-1}{2} \rfloor - 1$. Let's consider a 2-colouring of $K_{b,b}$, where $b = m + \lfloor \frac{n-1}{2} \rfloor - 1$. Let the independent sets of $K_{b,b}$ be A and B. Consider a maximal red kP_2, $k \leq n-1$. W.l.o.g. say it is made by the first k vertices in A and the first k vertices of B. Maximality forces a blue $K_{b-k,b-k}$ (composed by the last $b-k$ vertices of A and the last $b-k$ vertices of B) and so a blue S_{b-k+1}. Maximality of the red kP_2 also forces at least one of the two vertices of a P_2 in the upper sets to be joined with only blue edges to the lower sets. Therefore there exists a vertex in the lower sets with $b - k + \lceil \frac{k}{2} \rceil = b - \lfloor \frac{k}{2} \rfloor$ blue edges attached on it, i.e. at least a blue $S_{b-\lfloor \frac{k}{2} \rfloor + 1}$. As $k \leq n-1$ we get at least a blue $S_{b-\lfloor \frac{n-1}{2} \rfloor + 1} = S_m$. We will now show an upper bound revealing that:

$$R_b(S_m, nP_2) > m + \lfloor \frac{n-1}{2} \rfloor - 2.$$

Let $c = m + \lfloor \frac{n-1}{2} \rfloor - 2$. We consider the following 2-colouring of $K_{c,c}$ (see also Figure 8.12):

- Let the independent sets of $K_{c,c}$ be A and B.
- Colour the edges joining the first $m-2$ vertices of A with the first $m-2$ vertices of B blue.
- Colour the rest of the edges red.

□

The solution to the bipartite tree vs stripes problem is an immediate consequence of the following theorem.

Theorem 8.4.9 $R_b(T_m, nP_2) = max(n, \lceil \frac{m+n-1}{2} \rceil)$.

Proof We will first show that $R_b(T_m, nP_2) \leq max(n, \lceil \frac{m+n-1}{2} \rceil)$. We consider a 2-colouring of $K_{b,b}$, where $b = \lceil \frac{m}{2} \rceil + n - 1$. Let the independent

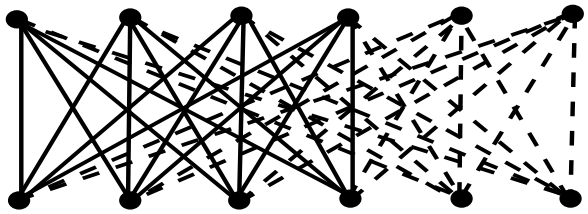


Figure 8.12: A 2 colouring of $K_{6,6}$ without a blue (continuous line) S_6 or a red (dashed line) $5P_2$

sets of $K_{b,b}$ be A and B. Consider a maximal red kP_2. W.l.o.g. say it is made by the first k vertices in A and the first k vertices of B. If $k = b$ then we get a red nP_2 otherwise $0 < k < b$ and maximality forces a blue $K_{b-k,b-k}$ (composed by the last $b-k$ vertices of A and the last $b-k$ vertices of B) and so a blue T_{2b-2k}. Therefore there exists at least one blue $T_{2b-2n+2}$ in our graph, i.e. at least one blue T_m. Maximality of the red kP_2 also forces at least one of the two vertices of a P_2 in the first sets of A and B to be joined with only blue edges to the lower sets. This forces the appearance of at least a blue $T_{2(b-k)+k} = T_{2b-k}$ composed by the vertices of the last sets of A and B and at least k vertices from the upper sets of A and B, i.e at least a blue $T_{2\lceil\frac{m+n-1}{2}\rceil-n+1}$ (which means a blue T_m if $m+n-1$ is even or a blue T_{m+1} if $m+n-1$ is odd).

We will now show that $R_b(T_m, nP_2) > \lceil\frac{m+n-1}{2}\rceil - 1$. Let $c = \lceil\frac{m+n-1}{2}\rceil - 1$. Consider the following 2-colouring of $K_{c,c}$ (similar to the colouring given in Figure 8.12):

- Let the independent sets of $K_{c,c}$ be A and B.
- Colour the edges joining the first $\lfloor\frac{m-1}{2}\rfloor$ vertices of A with the first $\lceil\frac{m-1}{2}\rceil$ vertices of B blue.
- Colour the rest of the edges red.

The lower bound construction for the case $n > \lceil\frac{m+n-1}{2}\rceil$ is a red $K_{n-1,n-1}$.

□

The solution for the bipartite Ramsey tree vs star problem is broken into several lemmas. We first establish a general upper bound, which can be achieved in certain cases, and then we provide a solution for the rest of the cases.

The following lemma gives us the general upper bound for this problem.

Lemma 8.4.10 $R_b(T_m, S_n) \leq n + \lfloor \frac{m-1}{2} \rfloor - 1$.

Proof Consider a 2-colouring of $K_{b,b}$, where $b = n + \lfloor \frac{m-1}{2} \rfloor - 1$. Let the independent sets of $K_{b,b}$ be A and B. Number the vertices in each set from 1 to b. Consider a maximal blue T_k, $k \leq m-1$. W.l.o.g. say it is made by the first x vertices in A and the first $k-x$ vertices of B. If $k = 2b$ we get a red $T_{2b} = T_{2(n+\lfloor \frac{m-1}{2} \rfloor - 1)}$ (i.e. at least a T_m), otherwise maximality forces a red $K_{b-x,k-x}$ (composed by the last $b-x$ vertices of A and the first $k-x$ vertices of B) and a red $K_{x,b+x-k}$ (composed by the first x vertices of A and the last $b+x-k$ vertices of B) and so a red S_{b-x+1}, a red S_{k-x+1}, a red S_{x+1} and a red $S_{b+x-k+1}$. The red S_{b-x+1} and the red $S_{b+x-k+1}$ guarantee at least a red $S_{b-\lfloor \frac{m-1}{2} \rfloor + 1} = S_n$. □

The above upper bound can be achieved in certain cases, as the next lemma shows.

Lemma 8.4.11 $R_b(T_m, S_n) > n + \lfloor \frac{m-1}{2} \rfloor - 2$, *if* $(n + \lfloor \frac{m-1}{2} \rfloor - 2) = x\lfloor \frac{m-1}{2} \rfloor + y(m-1)$ *where* $\{x, y\} \in \mathbb{Z}^*$.

Proof Let $b = n + \lfloor \frac{m-1}{2} \rfloor - 2$. Consider the following 2-colouring of $K_{b,b}$ (see also Figure 8.13):

- Let the independent sets of $K_{b,b}$ be A and B.
- Number the vertices in each set from 1 to b.
- Colour the edges joining the first $\lfloor \frac{m-1}{2} \rfloor$ vertices of A with the first $\lceil \frac{m-1}{2} \rceil$ vertices of B blue.

- Colour the edges joining the next $\lceil \frac{m-1}{2} \rceil$ vertices of A with the next $\lfloor \frac{m-1}{2} \rfloor$ vertices of B blue.
- Repeat colouring in this manner until we colour $2y$ sets.
- Colour the edges joining the next $\lfloor \frac{m-1}{2} \rfloor$ vertices of A with the next $\lfloor \frac{m-1}{2} \rfloor$ vertices of B blue.
- Repeat colouring in this manner until we reach the end of set A.
- Colour the rest of the edges red.

□

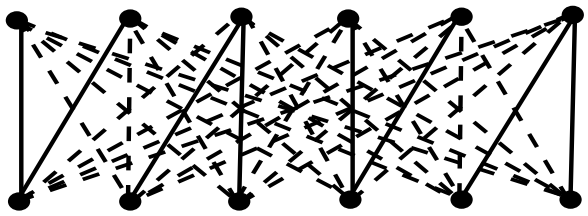


Figure 8.13: A 2 colouring of $K_{6,6}$ without a blue (continuous line) T_4 or a red (dashed line) S_7

The following theorem summarizes the above lemmas and provides us with the solution for the rest of the cases.

Theorem 8.4.12 *$R_b(T_m, S_n) = n + \lfloor \frac{m-1}{2} \rfloor - 1 - i$, where $i \in \mathbb{Z}^*$ and $\{y, x_0, \ldots, x_i\} \in \mathbb{Z}^*$ such that i is minimum and $(n + \lfloor \frac{m-1}{2} \rfloor - 2 - i) = y(m-1) + \sum_{j=0}^{i} x_j(\lfloor \frac{m-1}{2} \rfloor - j)$.*

Proof If $i = 0$ the theorem is an immediate consequence of Lemmas 8.4.10 and 8.4.11. The case of $i = 1$ implies that $n + \lfloor \frac{m-1}{2} \rfloor - 2$ is not a sum of $\lfloor \frac{m-1}{2} \rfloor$ and $m-1$ terms and we cannot use the same colouring as in Lemma 8.4.11. Any other colouring will lead to the creation of either a blue T_m or a red S_n and so $R_b(T_m, S_n) \leq n + \lfloor \frac{m-1}{2} \rfloor - 2 - 1$. Let $n + \lfloor \frac{m-1}{2} \rfloor - 2 - i = y(m-1) + \sum_{j=0}^{1} x_j(\lfloor \frac{m-1}{2} \rfloor - j)$. The following

colouring proves that $R_b(T_m, S_n) > n + \lfloor \frac{m-1}{2} \rfloor - 3$.

Consider the following 2-colouring of $K_{b,b}$, where $b = n + \lfloor \frac{m-1}{2} \rfloor - 3$:

- Let the independent sets of $K_{b,b}$ be A and B.
- Number the vertices in each set from 1 to b.
- Colour the edges joining the first $\lfloor \frac{m-1}{2} \rfloor$ vertices of A with the first $\lceil \frac{m-1}{2} \rceil$ vertices of B blue.
- Colour the edges joining the next $\lceil \frac{m-1}{2} \rceil$ vertices of A with the next $\lfloor \frac{m-1}{2} \rfloor$ vertices of B blue.
- Repeat colouring in this manner until we colour y sets.
- Colour the edges joining the next $\lfloor \frac{m-1}{2} \rfloor$ vertices of A with the next $\lfloor \frac{m-1}{2} \rfloor$ vertices of B blue.
- Repeat colouring in this manner until we colour x_0 sets.
- Colour the edges joining the next $\lfloor \frac{m-1}{2} \rfloor - 1$ vertices of A with the next $\lfloor \frac{m-1}{2} \rfloor - 1$ vertices of B blue.
- Repeat colouring in this manner until we colour x_1 sets.
- Colour the rest of the edges red.

Following similar arguments we can prove the statement for all possible values of i. □

Part V

Conclusion

Discrete structures such as graphs and words are an essential element in the field of Computer Science. In what follows we summarize our results and provide the reader with a list of open problems.

This thesis contains results on characteristics and regularities and words. Some computational results were shown:

- a linear-time algorithm for computing the minimal left-seed array of x [64]
- a linear-time solution for computing the maximal left-seed array of x [64]
- an $\mathcal{O}(n \log n)$ time algorithm for computing the minimal right-seed array of x [58]
- a linear-time solution for computing the maximal right-seed array of x [58]
- a linear time algorithm for the identification of all overlapping factors of a word. [52]
- a linear time algorithm that identifies all the repeating subtrees in a tree using a string representation of the tree[56, 55]

The frequency of appearance of regularities in words was also investigated:

- some bounds for the average number of powers with exponent r in a word were given, which were then extended for runs and palindromes as well as for some of their abelian varieties (abelian squares and cubes) [51]
- it was shown that a binary word of length n has almost surely $O(n^{\frac{3}{2}})$ abelian squares [51]

- we proved that a word has $O(n)$ seeds on average. [50]
- we showed that the maximum number of distinct seeds in a word is between $\frac{1}{6}n^2+o(n^2)$ and $\frac{1}{4}n^2+o(n^2)$ and we revealed some properties for the structure of an extremal word for the last case. [50]
- we provided some bounds on the maximum number of distinct overlapping factors in a word [48].

The appearance of abelian regularities in words was also investigated:

- two $O(n^2)$ algorithms for the computation of all abelian periods of a string x [60]
- an $O(n \log n)$ algorithm for the computation of all weak abelian periods of a word [60]
- a linear time algorithm for the computation of all abelian borders of a word x [49]
- an algorithm that finds the shortest abelian border of a non-abelian-border-free binary word in time $\Theta(\sqrt{n})$ on the average [53]
- it was proved that the average length of abelian borders of a word x is $\frac{n}{2}$, if it exists [49]
- it is proved that a binary word of length n has $\Theta(\sqrt{n})$ abelian borders on average [49]
- we investigated the number of binary words whose shortest abelian border has a given length, by identifying relations with Dyck words [53]
- we gave some bounds on the number of abelian border-free words of a given length and on the number of abelian words of a given length that have at least one abelian border [49]

The appearance of regularities in special words was also investigated:

- We identified all overlapping factors, left/right seeds, covers and seeds of a Fibonacci string as well as all covers of a circular Fibonacci string. [62, 63]

- we gave further comments on the number of distinct seeds in Fibonacci strings [63].

- we identified all borders, we give an elementary analysis of factors of the Padovan word, we identify all covers of Padovan words and we give some comments regarding the squares and cubes in a Padovan word.

- We also commented on the appearance of abelian borders in Fibonacci and Thue-Morse words [49].

We have also used string representations of trees to solve the problem of finding all repeating subtrees in a tree in linear time [56, 55].

Graphs arise in many areas of mathematics and computer science having applications in many other fields as well. Extremal graph theory problems usually ask for the maximum or minimum size or order of a graph having certain characteristics. Such questions are often quite natural in the construction of networks or circuits. Throughout this thesis we investigated extremal graph theory problems on special types of graphs:

- we considered the degree/diameter problem on trees, special types of trees such as Cayley trees, caterpillars, lobsters, banana trees and firecracker trees, as well as for tree-like structures such as pseudotrees, giving the extremal numbers and constructions [68].

- we considered the EX-problem for planar graphs and special types of planar graphs such as pseudotrees, cacti, Halin, generalized Halin

graphs and graphs lying in an infinite square grid, giving the extremal numbers and some constructions [66]. We also considered the problem on bipartite graphs. We gave the extremal cases for a bipartite graph of low order and girth, where $1 \leq n \leq 20$ and $4 \leq t \leq 10$, except for a few instances of the problem. We use these results to obtain the extremal cases for bipartite graphs of high girth and for the problem in the general case.

- we considered special cases of the bipartite Ramsey problem. More specifically we investigated the appearance of simpler monochromatic graphs such as stripes, stars and trees under a 2-colouring of the edges of a bipartite graph. We gave the Ramsey numbers $R_b(mP_2, nP_2)$, $R_b(T_m, T_n)$, $R_b(S_m, nP_2)$, $R_b(T_m, nP_2)$ and $R_b(S_m, T_n)$ [67].

Finally, to conclude we give a list of open problems in the area:

Problem 10 *The maximum number of distinct runs in a word of length n lies between $0.944575712n$ and $1.048n$. Can better bounds be found?*

Problem 11 *The maximum number of distinct runs in a word of length n lies between $0.406n$ and $0.5n$. Can better bounds be found?*

Problem 12 *The maximum number of distinct squares in a word of length n lies between $n - o(n)$ and $2n - \Theta(logn)$. Can better bounds be found?*

Problem 13 *The maximum number of distinct cubes in a word of length n lies between $\frac{n}{2}$ and $\frac{4n}{5}$. Can better bounds be found?*

Problem 14 *The maximum number of distinct seeds in a word of length n lies between $\frac{n^2}{6} + o(n^2)$ and $\frac{n^2}{4} + o(n^2)$. Can better bounds be found?*

Problem 15 *The maximum number of distinct overlapping factors in a word x lies between $\frac{5n^2}{48}$ and $\frac{n^2-n}{4}$. Can better bounds be found?*

Problem 16 *Is there an $o(n \log n)$ algorithm that can construct the right-seed array of a string?*

Problem 17 *Is there an $o(n^2)$ algorithm that identifies all abelian periods of a string?*

Problem 18 *Is there an $o(n \log n)$ algorithm that identifies all weak abelian periods of a string?*

Problem 19 *Is there an $o(n^2)$ algorithm that can construct the abelian border array of a string?*

Problem 20 *Is there an $o(n \log n)$ algorithm that identifies all weak abelian periods of a string?*

Problem 21 *The number of binary words of length n with no abelian borders, $|S_n|$, is $\Omega((\frac{3+\sqrt{13}}{2})^{\frac{n}{2}})$. Can better bounds be found?*

Problem 22 *The number of binary words of length n with at least one abelian borders, $|S'_n|$, lies between $\frac{2}{3} \cdot 2^n - \frac{2}{3}$ and 2^n, when n is even, and between $\frac{1}{3} \cdot 2^n - \frac{2}{3}$ and 2^n, when n is odd. Can better bounds be found?*

Problem 23 *Which factors of P_n are runs?*

Problem 24 *Which factors of P_n are abelian powers?*

Problem 25 *Which are the abelian periods of P_n?*

Problem 26 *What is the maximum number of vertices $n_{d,k}$ that can be contained in a cactus graph of maximum degree d and diameter at most k?*

Problem 27 *What is the maximum number of vertices $n_{d,k}$ that can be contained in a block graph of maximum degree d and diameter at most k?*

Problem 28 *The maximum number of edges in a bipartite graph of girth at least $t+1$ is at least $\lfloor \frac{2n-t-3}{t-1} \rfloor + n - 1$, when t is even and $t \geq 6$, and at least $\lfloor \frac{2n-t-4}{t-2} \rfloor + n - 1$, when t is odd and $t \geq 5$. Can better bounds be found?*

Problem 29 *What is the value of $R_b(C_m, C_n)$?*

Problem 30 *What is the value of $R_b(C_m, S_n)$?*

Problem 31 *What is the value of $R_b(C_m, P_n)$?*

Problem 32 *What is the value of $R_b(C_m, nP_2)$?*

Problem 33 *What is the value of $R_b(C_m, T_n)$?*

Problem 34 (Validity Problem) *Let A be an integer array of length n. Decide if A is the minimal left (resp. right) seed array of some string.*

Problem 35 (Construction Problem) *Let A be an integer array of length n. When A is a valid minimal left-seed (resp. right-seed) array, exhibit a string over an unbounded alphabet whose minimal left-seed (resp. right-seed) array is A.*

Bibliography

[1] E. Abajo, C. Balbuena, and A. Diánez. New families of graphs without short cycles and large size. *Discrete Applied Mathematics*, 158(11):1127–1135, 2010.

[2] E. Abajo and A. Diánez. Size of graphs with high girth. *Electronic Notes in Discrete Mathematics*, 29:179–183, 2007.

[3] E. Abajo and A. Dianez. Exact values of ex (ν;{C3, C4,..., Cn}). *Discrete Applied Mathematics*, 158(17):1869–1878, 2010.

[4] E. Abajo and A. Diánez. Graphs with maximum size and lower bounded girth. *Applied Mathematics Letters*, 2011.

[5] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: principles, techniques, and tools*. Addison Wesley, 2nd edition, 2006.

[6] J. Allouche and J. Shallit. *Automatic sequences: theory, applications, generalizations*. Cambridge University Press, 2003.

[7] A. Apostolico and D. Breslauer. Of periods, quasiperiods, repetitions and covers. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, volume 1261 of *Lecture Notes in Computer Science*, pages 236–248. Springer, 1997.

[8] A. Apostolico and M. Crochemore. String pattern matching for a deluge survival kit. In J. Abello, P. M. Pardalos, and M. G. C.

Resende, editors, *Handbook of massive data sets*, pages 151–194. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[9] A. Apostolico and A. Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.*, 119(2):247–265, 1993.

[10] A. Apostolico, M. Farach, and C. S. Iliopoulos. Optimal superprimitivity testing for strings. *Information Processing Letters*, 39:17–20, 1991.

[11] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22(3):297–315, 1983.

[12] A. Atkin and D. Bernstein. Prime sieves using binary quadratic forms. *Mathematics of Computation*, 73(246):1023–1030, 2004.

[13] S. Avgustinovich, A. Glen, B. Halldórsson, and S. Kitaev. On shortest crucial words avoiding abelian powers. *Discrete Applied Mathematics*, 158(6):605–607, 2010.

[14] S. Avgustinovich, J. Karhumaki, and S. Puzynina. On abelian versions of critical factorization theorem. *Proceedings of the 13th Mons. Theoretical Computer Science Days*, 2010.

[15] C. Balbuena, M. Cera, A. Diánez, and P. García-Vázquez. On the girth of extremal graphs without shortest cycles. *Discrete Mathematics*, 308(23):5682–5690, 2008.

[16] C. Balbuena, P. Garcia-Vazquez, X. Marcote, and J. Valenzuela. Extremal bipartite graphs with high girth. *Ars Combinatoria*, 83:3, 2007.

[17] C. Balbuena, T. Jiang, Y. Lin, X. Marcote, and M. Miller. A lower bound on the order of regular graphs with given girth pair. *Journal of Graph Theory*, 55(2):153–163, 2007.

[18] E. Bannai and T. Ito. On finite moore graphs. *Journal of the Faculty of Science, University of Tokyo: Mathematics*, 20:191, 1973.

[19] E. Bannai and T. Ito. Regular graphs with excess one. *Discrete Mathematics*, 37(2-3):147–158, 1981.

[20] R. Bar-Yehuda and T. Etzion. Connections between two cycles–a new design of dense processor interconnection networks. *Discrete Applied Mathematics*, 37:29–43, 1992.

[21] E. Baskoro, M. Miller, J. Plesnık, and Š. Znám. Regular digraphs of diameter 2 and maximum order. *Australasian Journal of Combinatorics*, 9:291–306, 1994.

[22] L. Beineke and A. Schwenk. On a bipartite form of the ramsey problem. In *Proceedings of the Fifth British Combinatorial Conference (Univ. Aberdeen, Aberdeen, 1975)*, pages 17–22, 1975.

[23] A. Benjamin, A. Benjamin, and J. Quinn. *Proofs that really count: the art of combinatorial proof*. The Mathematical Association of America, 2003.

[24] J. Berstel. Fibonacci words-a survey. *The book of L*, pages 13–27, 1986.

[25] J. Berstel. *Combinatorics on Words: Christoffel words and repetitions in words*, volume 27. American Mathematical Society, 2009.

[26] F. Blanchet-Sadri, J. Kim, R. Mercaş, W. Severa, and S. Simmons. Abelian square-free partial words. *Language and Automata Theory and Applications*, pages 94–105, 2010.

[27] F. Blanchet-Sadri and R. Mercas. A note on the number of squares in a partial word with one hole. *RAIRO- Theoretical Informatics and Applications*, 43(4):767–774, 2009.

[28] F. Blanchet-Sadri, A. Tebbe, and A. Veprauskas. Fine and Wilf's theorem for abelian periods in partial words. *Proceedings of the 13th Mons Theoretical Computer Science Days, Amiens, France*, 2010.

[29] B. Bollobás. *Extremal graph theory*. Dover Publications, 1978.

[30] J. A. Bondy. Pancyclic graphs i. *Journal of Combinatorial Theory, Series B*, 11(1):80–84, 1971.

[31] J. A. Bondy and L. Lovasz. Lengths of cycles in Halin graphs. *Journal of graph theory*, 9(3):397–410, 1985.

[32] A. Boyd. Bounds for the catalan numbers. *Fibonacci Quarterly*, 1992.

[33] D. Breslauer. An on-line string superprimitivity test. *Information Processing Letters*, 44(6):345–347, 1992.

[34] D. Breslauer. Testing string superprimitivity in parallel. *Information Processing Letters*, 49(5):235–241, 1994.

[35] W. Bridges and S. Toueg. On the impossibility of directed moore graphs. *Journal of Combinatorial theory, series B*, 29(3):339–341, 1980.

[36] G. Brodal and C. Pedersen. Finding maximal quasiperiodicities in strings. In *Combinatorial Pattern Matching*, pages 397–411. Springer, 2000.

[37] D. Bryant and H. Fu. C4-saturated bipartite graphs. *Discrete Mathematics*, 259(1-3):263–268, 2002.

[38] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On table arrangements, scrabble freaks, and jumbled pattern matching. In *Fun with Algorithms*, pages 89–101. Springer, 2010.

[39] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. Algorithms for jumbled pattern matching in strings. *International Journal of Foundations of Computer Science*, 2011.

[40] D. Buset. Maximal cubic graphs with diameter 4. *Discrete Applied Mathematics*, 101(1-3):53–61, 2000.

[41] D. Callan. Card deals, lattice paths, abelian words and combinatorial identities. *ArXiv e-prints*, Dec. 2008. arXiv:0812.4784v1.

[42] W. Carnielli and E. Carmelo. $k_{2,2} - k_{1,n}$ and $k_{2,n} - k_{2,n}$ bipartite ramsey numbers. *Discrete Mathematics*, 223(1):83–92, 2000.

[43] W. Carnielli and E. Monte Carmelo. On the ramsey problem for multicolor bipartite graphs. *Advances in Applied Mathematics*, 22(1):48–59, 1999.

[44] Y. Caro and C. Rousseau. Asymptotic bounds for bipartite ramsey numbers. *Journal of Combinatorics*, 8(1):17–17, 2001.

[45] J. Cassaigne, G. Richomme, K. Saari, and L. Q. Zamboni. Avoiding abelian powers in binary words with bounded abelian complexity. *International Journal of Foundations of Computer Science*, 22(4):905–920, 2011.

[46] M. Cera, A. Diánez, and P. Vázquez. Structure of the extremal family ex (n; tkp). *Electronic Notes in Discrete Mathematics*, 10:72–74, 2001.

[47] G.-H. Chen, J.-J. Hong, and H.-I. Lu. An optimal algorithm for online square detection. In *Combinatorial Pattern Matching*, pages 280–287. Springer, 2005.

[48] M. Christodoulakis and M. Christou. Abelian concepts in words-a review. *Festschrift for Bořivoj Melichar, Prague Stringology Conference*, 2012.

[49] M. Christodoulakis, M. Christou, M. Crochemore, and C. S. Iliopoulos. Abelian borders in words. *Fundamenta Informaticae*, 2013. (accepted).

[50] M. Christodoulakis, M. Christou, M. Crochemore, and C. S. Iliopoulos. On the appearance of seeds in words. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 2013.

[51] M. Christodoulakis, M. Christou, M. Crochemore, and C. S. Iliopoulos. On the average number of regularities in a word. *Theoretical Computer Science*, 2013.

[52] M. Christodoulakis, M. Christou, M. Crochemore, and C. S. Iliopoulos. Overlapping factors in words. *Australasian Journal of Combinatorics*, 2013.

[53] M. Christodoulakis, M. Christou, M. Crochemore, and C. S. Iliopoulos. Abelian borders in binary words. *Discrete Applied Mathematics*, 171:141–146, 2014.

[54] M. Christodoulakis, C. Iliopoulos, K. Park, and J. Sim. Approximate seeds of strings. *Journal of Automata, Languages and Combinatorics*, 10(5/6):609–626, 2005.

[55] M. Christou, M. Crochemore, T. Flouri, C. Iliopoulos, J. Janousek, B. Melichar, and S. Pissis. Computing all subtree repeats in ordered trees. *Information Processing Letters*, 112(24):958–962, 2012.

[56] M. Christou, M. Crochemore, T. Flouri, C. S. Iliopoulos, J. Janoušek, B. Melichar, and S. P. Pissis. Computing all subtree repeats in ordered ranked trees. In *String Processing and Information Retrieval*, pages 338–343. Springer, 2011.

[57] M. Christou, M. Crochemore, O. Guth, C. Iliopoulos, and S. Pissis. On the right-seed array of a string. *Computing and Combinatorics*, pages 492–502, 2011.

[58] M. Christou, M. Crochemore, O. Guth, C. S. Iliopoulos, and S. P. Pissis. On the right-seed array of a string. In B. Fu and D.-Z. Du, editors, *Proceedings of the seventeenth annual International Computing and Combinatorics Conference (COCOON 2011)*, volume 6842 of *Lecture Notes in Computer Science*, pages 492–502, USA, 2011. Springer.

[59] M. Christou, M. Crochemore, O. Guth, C. S. Iliopoulos, and S. P. Pissis. On left and right seeds of a string. *Journal of Discrete Algorithms*, 2012.

[60] M. Christou, M. Crochemore, and C. Iliopoulos. Identifying all abelian periods of a string in quadratic time and relevant problems. *International Journal of Foundations of Computer Science*, 2012. (accepted).

[61] M. Christou, M. Crochemore, C. Iliopoulos, M. Kubica, S. Pissis, J. Radoszewski, W. Rytter, B. Szreder, and T. Waleń. Efficient seeds computation revisited. In *Combinatorial Pattern Matching*, pages 350–363. Springer, 2011.

[62] M. Christou, M. Crochemore, and C. S. Iliopoulos. Quasiperiodicities in Fibonacci strings. In *Local Proceedings of International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2012.

[63] M. Christou, M. Crochemore, and C. S. Iliopoulos. Quasiperiodicities in Fibonacci strings. *Ars Combinatoria*, 2012. (accepted).

[64] M. Christou, M. Crochemore, C. S. Iliopoulos, M. Kubica, S. P. Pissis, J. Radoszewski, W. Rytter, B. Szreder, and T. Walen. Efficient seed computation revisited. In R. Giancarlo and G. Manzini, editors, *Proceedings of the twenty-second annual Symposium on Combinatorial Pattern Matching (CPM 2011)*, volume 6661 of *Lecture Notes in Computer Science*, pages 350–363, Italy, 2011. Springer.

[65] M. Christou, M. Crochemore, C. S. Iliopoulos, M. Kubica, S. P. Pissis, J. Radoszewski, W. Rytter, B. Szreder, and T. Waleń. Efficient seed computation revisited. *Theoretical Computer Science*, 2012.

[66] M. Christou, C. S. Iliopoulos, and M. Miller. Maximizing the size of planar graphs under girth constraints. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 2012. (accepted).

[67] M. Christou, C. S. Iliopoulos, and M. Miller. Bipartite ramsey numbers involving stars, stripes and trees. *Electronic Journal of Graph Theory and Applications*, 2013.

[68] M. Christou, C. S. Iliopoulos, and M. Miller. Degree/diameter problem for trees and pseudotrees. *AKCE International Journal of Graphs and Combinatorics*, 2013.

[69] W. Chuan and H. Ho. Locating factors of the infinite fibonacci word. *Theoretical computer science*, 349(3):429–442, 2005.

[70] F. Chung and R. Graham. On multicolor ramsey numbers for complete bipartite graphs. *Journal of Combinatorial Theory, Series B*, 18:164–169, 1975.

[71] F. Cicalese, G. Fici, and Z. Lipták. Searching for jumbled patterns in strings. In *Proceedings of the Prague Stringology Conference (PSC 2009)*, 2009.

[72] M. Conder and R. Nedela. A more detailed classification of symmetric cubic graphs. *preprint*, 2006.

[73] D. Conlon. A new upper bound for the bipartite ramsey problem. *Journal of Graph Theory*, 58(4):351–356, 2008.

[74] S. Constantinescu and L. Ilie. Fine and Wilf's theorem for abelian periods. *Bulletin of the EATCS*, 89:167–170, 2006.

[75] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.

[76] M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986.

[77] M. Crochemore, S. Fazekas, C. Iliopoulos, and I. Jayasekera. Bounds on powers in strings. In M. Ito and M. Toyama, editors, *Developments in Language Theory*, volume 5257 of *Lecture Notes in Computer Science*, pages 206–215. Springer, 2008.

[78] M. Crochemore, S. Fazekas, C. Iliopoulos, and I. Jayasekera. Number of occurrences of powers in strings. *International Journal of Foundations of Computer Science*, 21(4):535–547, 2010.

[79] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, New York, NY, USA, 2007.

[80] M. Crochemore, L. Ilie, and W. Rytter. Repetitions in strings: Algorithms and combinatorics. *Theoretical Computer Science*, 410(50):5227–5235, 2009.

[81] M. Crochemore, L. Ilie, and L. Tinta. Towards a solution to the runs conjecture. In P. Ferragina and G. M. Landau, editors, *Combinatorial Pattern Matching*, volume 5029 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2008.

[82] M. Crochemore, L. Ilie, and L. Tinta. The "runs" conjecture. *Theoretical Computer Science*, 412(27):2931–2941, 2011.

[83] M. Crochemore, C. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. On the maximal number of cubic runs in a string. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, 2010.

[84] M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, K. Stencel, and T. Walen. New simple efficient algorithms computing powers and runs in strings. In *Prague Stringology Conference*, pages 138–149, 2010.

[85] L. Cummings, D. Moore, and J. Karhumäki. Borders of fibonacci strings. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 20:81–88, 1996.

[86] L. J. Cummings and W. F. Smyth. Weak repetitions in strings. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 24:33–48, 1997.

[87] J. Currie and A. Aberkane. A cyclic binary morphism avoiding abelian fourth powers. *Theoretical Computer Science*, 410(1):44–52, 2009.

[88] J. Currie and T. Visentin. On abelian 2-avoidable binary patterns. *Acta Informatica*, 43(8):521–533, 2007.

[89] E. Curtin. Cubic cayley graphs with small diameter. *Discrete Mathematics and Theoretical Computer Science*, 4:123–132, 2001.

[90] R. Damerell. On Moore graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 74(02):227–236, 1973.

[91] D. De Caen and L. Székely. The maximum size of 4-and 6-cycle free bipartite graphs on m, n vertices. *Sets, graphs and numbers: a birthday salute to Vera T. Sós and András Hajnal*, 60(60):135, 1992.

[92] D. De Caen and L. Székely. On dense bipartite graphs of girth eight and upper bounds for certain configurations in planar point–line systems. *Journal of Combinatorial Theory, Series A*, 77(2):268–278, 1997.

[93] C. Delorme, E. Flandrin, Y. Lin, M. Miller, and J. Ryan. On extremal graphs with bounded girth. *Electronic Notes in Discrete Mathematics*, 34:653–657, 2009.

[94] M. Domaratzki and N. Rampersad. Abelian primitive words. In G. Mauri and A. Leporati, editors, *Proceedings of the 15th International Conference on Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 204–215. Springer, 2011.

[95] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *Journal of ACM*, 27(4):758–771, 1980.

[96] D. Downing and J. Clark. *Statistics: The Easy Way*. Barron's Educational Series, 1997.

[97] X. Droubay. Palindromes in the fibonacci word. *Information Processing Letters*, 55(4):217–221, 1995.

[98] M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. *Journal of ACM*, 41(2):205–213, 1994.

[99] R. Dunlap. *The golden ratio and Fibonacci numbers*. World Scientific Pub Co Inc, 1997.

[100] T. Ejaz. *Abelian Pattern Matching in Strings*. PhD thesis, Technische Universität Dortmund, 2010.

[101] T. Ejaz, S. Rahmann, and J. Stoye. Online abelian pattern matching. Technical report, Technische Universität Dortmund, 2008.

[102] B. Elspas. Topological constraints on interconnection-limited logic. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 133–137. IEEE, 1964.

[103] P. Erdős, A. Sárközy, and V. T. Sós. On product representations of powers, i. *Eur. J. Comb.*, 16(6):567–588, Nov. 1995.

[104] P. Erdős. *Some unsolved problems*. Magyar Tudományos Akadémia Matematikai Kutató Intézete, 1961.

[105] P. Erdos. Some recent progress on extremal problems in graph theory. *Congressus Numerantium*, 14:3–14, 1975.

[106] P. Erdös, S. Fajtlowicz, and A. Hoffman. Maximum degree in graphs of diameter 2. *Networks*, 10:87–90, 1980.

[107] P. Erdös and H. Sachs. Reguläre graphen gegebener taillenweite mit minimaler knotenzahl. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 12:251–257, 1963.

[108] L. Eroh and O. Oellermann. Bipartite rainbow ramsey numbers. *Discrete Mathematics*, 277(1):57–72, 2004.

[109] G. Exoo. A bipartite ramsey number. *Graphs and Combinatorics*, 7(4):395–396, 1991.

[110] G. Exoo and R. Jajcay. Dynamic cage survey. *Electronic Journal of Combinatorics*, 15:16, 2008.

[111] R. Faudree and R. Schelp. Path-path ramsey-type numbers for the complete bipartite graph. *Journal of Combinatorial Theory, Series B*, 19(2):161–173, 1975.

[112] M. Fellows, P. Hell, and K. Seyffarth. Constructions of large planar networks with given degree and diameter. *Networks*, 32(4):275–281, 1998.

[113] G. Fici, A. Langiu, T. Lecroq, A. Lefebvre, F. Mignosi, and É. Prieur-Gaston. Abelian repetitions in sturmian words. In *Developments in Language Theory*, pages 227–238. Springer, 2013.

[114] G. Fici, T. Lecroq, A. Lefebvre, and É. Prieur-Gaston. Computing abelian periods in words. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference (PSC 2011)*, pages 184–196, Czech Technical University in Prague, Czech Republic, 2011.

[115] G. Fici, T. Lecroq, A. Lefebvre, É. Prieur-Gaston, and W. F. Smyth. Quasi-linear time computation of the abelian periods of a word. In *Stringology*, pages 103–110, 2012.

[116] P. Flajolet, P. Sipala, and J.-M. Steyaert. Analytic variations on the common subexpression problem. In M. S. Paterson, editor, *Proceedings of the seventeenth International Colloquium on Automata,*

Languages and Programming (ICALP 1990), volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990.

[117] A. Fraenkel and J. Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998.

[118] A. Fraenkel and J. Simpson. The exact number of squares in fibonacci words. *Theoretical Computer Science*, 218(1):95–106, 1999.

[119] A. Fraenkel, J. Simpson, and M. Paterson. On weak circular squares in binary words. In *Combinatorial Pattern Matching*, pages 76–82. Springer, 1997.

[120] A. Fraenkel, J. Simpson, and M. Paterson. On abelian circular squares in binary words. In *Paul Erdos and his Mathematics II*, volume 11 of *Bolyai Soc., Mathematical Studies*, pages 329–338, Budapest, 2002.

[121] F. Franek and M. Jiang. Crochemore's repetitions algorithm revisited: Computing runs. *International Journal of Foundations of Computer Science*, 23(02):389–401, 2012.

[122] F. Franek, M. Jiang, and C.-C. Weng. An improved version of the runs algorithm based on crochemoreâĂŹs partitioning algorithm. In *Proceedings of Prague Stringology Conference*, pages 98–105, 2011.

[123] F. Franek, Q. Yang, and J. Holub. An asymptotic lower bound for the maximal number of runs in a string. *International Journal of Foundations of Computer Science*, 19(1):195–203, 2008.

[124] E. Friedman and R. Pratt. New bounds for largest planar graphs with fixed maximum degree and diameter. *Preprint*, 1999.

[125] Z. Galil. Palindrome recognition in real time by a multitape turing machine. *Journal of Computer and System Sciences*, 16(2):140–157, 1978.

[126] P. Garcia-Vazquez, C. Balbuena, X. Marcote, and J. Valenzuela. On extremal bipartite graphs with high girth. *Electronic Notes in Discrete Mathematics*, 26:67–73, 2006.

[127] D. Garnick, Y. Kwong, and F. Lazebnik. Extremal graphs without three-cycles or four-cycles. *Journal of Graph Theory*, 17(5):633–645, 1993.

[128] D. Garnick and N. Nieuwejaar. Non-isomorphic extremal graphs without three-cycles or four-cycles. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 12:33–56, 1992.

[129] M. Giraud. Not so many runs in strings. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *Language and Automata Theory and Applications*, volume 5196 of *Lecture Notes in Computer Science*. Springer, 2008.

[130] W. Goddard, M. Henning, and O. Oellermann. Bipartite ramsey numbers and zarankiewicz numbers. *Discrete Mathematics*, 219:85–95, 2000.

[131] R. Grossi. On finding commong subtrees. *Theoretical Computer Science*, 108(2):345–356, 1993.

[132] R. Groult, É. Prieur, and G. Richomme. Counting distinct palindromes in a word in linear time. *Information Processing Letters*, 110(20):908–912, 2010.

[133] R. Groult and G. Richomme. Optimality of some algorithms to detect quasiperiodicities. *Theoretical Computer Science*, 411(34-36):3110–3122, 2010.

[134] S. Guillemot and F. Nicolas. Solving the maximum agreement subtree and the maximum compatible tree problems on many bounded degree trees. *CoRR*, abs/0802.0024, 2008.

[135] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

[136] D. Gusfield and J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004.

[137] A. Gyárfás and J. Lehel. A ramsey-type problem in directed and bipartite graphs. *Periodica Mathematica Hungarica*, 3(3):299–304, 1973.

[138] E. Györi. C_6-free bipartite graphs and product representation of squares. *Discrete Mathematics*, 165:371–375, 1997.

[139] V. Halava, T. Harju, T. Kärki, and P. Séébold. Overlap-freeness in infinite partial words. *Theoretical Computer Science*, 410(8-10):943–948, 2009.

[140] G. Hardy and E. Wright. *An introduction to the theory of numbers.* Oxford University Press, USA, 1979.

[141] J. Hattingh and M. Henning. Bipartite ramsey theory. *Utilitas Mathematica*, pages 217–230, 1998.

[142] J. Hattingh and M. Henning. Star-path bipartite ramsey numbers. *Discrete Mathematics*, 185(1):255–258, 1998.

[143] M. Henning and O. Oellermann. Bipartite ramsey theorems for multiple copies of $k_{2,2}$. *Utilitas Mathematica*, pages 23–24, 1998.

[144] S. Heubach and T. Mansour. *Combinatorics of compositions and words*. Chapman & Hall/CRC, 2009.

[145] A. Hoffman and R. Singleton. On Moore graphs with diameters 2 and 3. *IBM Journal of Research and Development*, 4(5):497–504, 1960.

[146] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *Journal of ACM*, 29(1):68–95, 1982.

[147] S. Hoory. The size of bipartite graphs with a given girth. *Journal of Combinatorial Theory, Series B*, 86(2):215–220, 2002.

[148] J. E. Hopcroft, J. D. Ullman, and A. Aho. The design and analysis of computer algorithms. *Reading, Addison-Wesley, London*, 1974, 1974.

[149] L. Ilie. A simple proof that a word of length n has at most $2n$ distinct squares. *Journal of Combinatorial Theory, Series A*, 112(1):163–164, 2005.

[150] L. Ilie. A note on the number of squares in a word. *Theoretical Computer Science*, 380(3):373–376, 2007.

[151] C. Iliopoulos, D. Moore, and W. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172(1-2):281–291, 1997.

[152] C. Iliopoulos, D. Moore, and W. Smyth. The covers of a circular Fibonacci string. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 26:227–236, 1998.

[153] C. Iliopoulos and K. Park. A work-time optimal algorithm for computing all string covers. *Theoretical Computer Science*, 164(1-2):299–310, 1996.

[154] C. S. Iliopoulos, D. W. G. Moore, and K. Park. Covering a string. *Algorithmica*, 16:289–297, Sept. 1996.

[155] C. S. Iliopoulos and L. Mouchard. Quasiperiodicity: From detection to normal forms. *Journal of Automata, Languages and Combinatorics*, 4(3):213–228, 1999.

[156] R. Irving. A bipartite ramsey problem and the zarankiewicz numbers. *Glasgow Mathematical Journal*, 19(01):13–26, 1978.

[157] J. Jansson and Z. Peng. Online and dynamic recognition of square-free strings. In *Mathematical Foundations of Computer Science 2005*, pages 520–531. Springer, 2005.

[158] J. Jansson and Z. Peng. Online and dynamic recognition of square-free strings. *International Journal of Foundations of Computer Science*, 18(02):401–414, 2007.

[159] J. Jeuring. Finding palindromes. In *SION Computing Science in the Netherlands*, pages 123–140, 1988.

[160] J. Jeuring. The derivation of on-line algorithms, with an application to finding palindromes. *Algorithmica*, 11(2):146–184, 1994.

[161] A. Karaman. Weak repetitions in sturmian strings. *Theoretical Computer Science*, 290(3):2137–2146, 2003.

[162] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the fourth annual ACM Symposium on Theory of Computing (STOC 1972)*, pages 125–136. ACM, 1972.

[163] W. Kautz. Bounds on directed (d, k) graphs. theory of cellular logic networks and machines. *AFCRL-687ndash; 0668 Final report*, pages 20–28, 1968.

[164] V. Keränen. Abelian squares are avoidable on 4 letters. *Automata, Languages and Programming*, pages 41–52, 1992.

[165] D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[166] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. A linear time algorithm for seeds computation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1095–1112. SIAM, 2012.

[167] R. Kolpakov and G. Kucherov. Maximal repetitions in words or how to find all squares in linear time. Technical report, Laboratoire lorrain de recherche en informatique et ses applications, 1998.

[168] R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Symposium on Foundations of Computer Science-FOCS*, volume 99, pages 596–604, 1999.

[169] R. Kolpakov, G. Kucherov, et al. On maximal repetitions in words. *Journal on Discrete Algorithms*, 1(1):159–186, 2000.

[170] S. R. Kosaraju. Efficient tree pattern matching. In *Proceedings of the thirtieth annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 178–183. IEEE Computer Society, 1989.

[171] S. R. Kosaraju. Computation of squares in a string. In *Combinatorial Pattern Matching*, pages 146–150. Springer, 1994.

[172] M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. On the maximal number of cubic subwords in a string. In J. Fiala, J. Kra-

tochvíl, and M. Miller, editors, *International Workshop on Combinatorial Algorithms*, volume 5874 of *Lecture Notes in Computer Science*. Springer, 2009.

[173] T. Kuboyama. *Matching and learning in trees*. PhD thesis, University of Tokyo, 2007.

[174] G. Kucherov, P. Ochem, and M. Rao. How many square occurrences must a binary sequence contain? *Journal of Combinatorics*, 10(1):12, 2003.

[175] K. Kurosawa and S. Tsujii. Considerations on diameter of communication networks. *Electronics and Communications in Japan (Part I: Communications)*, 64(4):37–45, 1981.

[176] Y. Kwong, D. Garnick, and F. Lazebnik. Extremal graphs without three-cycles or four-cycles. *Journal of Graph Theory*, 17(5):633–645, 1993.

[177] T. Lam. Graphs without cycles of even length. *Bulletin of the Australian Mathematical Society*, 63(03):435–440, 2001.

[178] T. Lam. A result on 2k-cycle free bipartite graphs. *Australasian Journal of Combinatorics*, 32:163–170, 2005.

[179] F. Lazebnik, V. Ustimenko, and A. Woldar. New constructions of bipartite graphs on m, n, vertices with many edges and without small cycles. *Journal of Combinatorial Theory, Series B*, 61(1):111–117, 1994.

[180] H.-F. Leung, Z. Peng, and H.-F. Ting. An efficient online algorithm for square detection. In *Computing and Combinatorics*, pages 432–439. Springer, 2004.

[181] Y. Li and W. F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32(1):95–106, 2002.

[182] Q. Lin and Y. Li. Bipartite ramsey numbers involving large $k_{n,n}$. *European Journal of Combinatorics*, 30(4):923–928, 2009.

[183] V. Longani. Some bipartite ramsey numbers. *Southeast Asian Bulletin of Mathematics*, 26(4):583–592, 2003.

[184] M. Lothaire, editor. *Algebraic Combinatorics on Words*. Cambridge University Press, 2001.

[185] M. Lothaire, editor. *Applied Combinatorics on Words*. Cambridge University Press, 2005.

[186] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989.

[187] M. G. Main and R. J. Lorentz. An o(n log n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

[188] M. G. Main and R. J. Lorentz. Linear time recognition of squarefree strings. *Combinatorial Algorithms on Words*, 12:271–278, 1985.

[189] G. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, July 1975.

[190] W. Mantel. Problem 28. *Wiskundige Opgaven*, 10(60-61):320, 1907.

[191] W. Matsubara, K. Kusano, H. Bannai, and A. Shinohara. A series of run-rich strings. In A. H. Dediu, A.-M. Ionescu, and C. Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 5457 of *Lecture Notes in Computer Science*. Springer, 2009.

[192] W. Matsubara, K. Kusano, A. Ishino, H. Bannai, and A. Shinohara. New lower bounds for the maximum number of runs in a string. In *Prague Stringology Conference*, volume 2008, pages 140–145, 2008.

[193] G. Mauri and G. Pavesi. Algorithms for pattern matching and discovery in RNA secondary structure. *Theoretical Computer Science*, 335(1):29–51, 2005.

[194] M. Miller and J. Širán. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic Journal of Combinatorics*, 61:1–61, 2005.

[195] S. Molodtsov. Largest graphs of diameter 2 and maximum degree 6. *Electronic Notes in Discrete Mathematics*, 21:365–366, 2005.

[196] D. Moore and W. Smyth. An optimal algorithm to compute all the covers of a string. *Information Processing Letters*, 50(5):239–246, 1994.

[197] D. Moore and W. F. Smyth. A correction to âĂIJan optimal algorithm to compute all the covers of a stringâĂİ. *Information Processing Letters*, 54(2):101–103, 1995.

[198] A. Naor and J. Verstraëte. A note on bipartite graphs without 2k-cycles. *Combinatorics, Probability and Computing*, 14(5-6):845–849, 2005.

[199] S. Neuwirth. The size of bipartite graphs with girth eight. *arXiv preprint math/0102210*, 2001.

[200] S. J. Pan and R. Lee. Looking for all palindromes in a string. In *The 23rd Workshop on Combinatorial Mathematics and Computation Theory*, 2006.

[201] P. Pleasants. Non-repetitive sequences. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 68, pages 267–274. Cambridge Univ Press, 1970.

[202] S. Puglisi, J. Simpson, and W. Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401(1-3):165–171, 2008.

[203] S. J. Puglisi and J. Simpson. The expected number of runs in a word. *Australasian Journal of Combinatorics*, 42:45–54, 2008.

[204] M. O. Rabin. Discovering repetitions in strings. *Combinatorial Algorithms on Words*, 12:279–288, 1985.

[205] S. Radziszowski. Small ramsey numbers. *The Electronic Journal of Combinatorics*, 1000(0):DS1–Aug, 2011.

[206] F. Ramsey. On a problem of formal logic. *Classic Papers in Combinatorics*, pages 1–24, 1987.

[207] L. Richmond and J. Shallit. Counting abelian squares. *The Electronic Journal of Combinatorics*, 16(1):R72, 2009.

[208] G. Richomme, K. Saari, and L. Zamboni. Standard words and abelian powers in sturmian words. In *Proceedings of the 12th Mons Theoretical Computer Science Days (Mons Days of Theoretical Computer Science)*, 2008.

[209] G. Richomme, K. Saari, and L. Zamboni. Balance and abelian complexity of the tribonacci word. *Advances in Applied Mathematics*, 45(2):212–231, 2010.

[210] K. F. Riley, M. P. Hobson, and S. J. Bence. *Mathematical methods for physics and engineering (3rd edition)*. Cambridge University Press, 2010.

[211] S. Rosema and R. Tijdeman. The tribonacci substitution. *Integers: Electronic Journal of Combinatorial Number Theory*, 5(3):A13, 2005.

[212] V. Rosta. Ramsey theory applications. *The Electronic Journal of Combinatorics*, pages 1–43, 2004.

[213] Z. Rui and S. Yongqi. The bipartite ramsey numbers b(c_2m; $k_{2,2}$). *The Electronic Journal of Combinatorics*, 18(P51):1, 2011.

[214] W. Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In B. Durand and W. Thomas, editors, *STACS*, volume 3884 of *Lecture Notes in Computer Science*. Springer, 2006.

[215] W. Rytter. The number of runs in a string. *Information and Computation*, 205(9):1459–1469, 2007.

[216] H. Sachs. Regular graphs with given girth and restricted circuits. *Journal of the London Mathematical Society*, 1(1):423–429, 1963.

[217] A. Samsonov and A. Shur. On abelian repetition threshold. *Proc. 13th Mons Days of Theoretical Computer Science. Univ. de Picardie Jules Verne, Amiens*, pages 1–11, 2010.

[218] G. N. Sárközy. Cycles in bipartite graphs and an application in number theory. *Journal of Graph Theory*, 19(3):323–331, 1995.

[219] P. Séébold. Overlap-free sequences. *Automata on Infinite Words*, pages 207–215, 1985.

[220] J. Simpson. Modified padovan words and the maximum number of runs in a word. *Australasian Journal of Combinatorics*, 46:129–145, 2010.

[221] A. Slisenko. Detection of periodicities and string-matching in real time. *Journal of Soviet Mathematics*, 22(3):1316–1387, 1983.

[222] A. O. e. Slisenko. Detection of periodicities and string-matching in real time. *Zapiski Nauchnykh Seminarov POMI*, 105:62–173, 1981.

[223] N. Sloane and S. Plouffe. *The encyclopedia of integer sequences*. Academic Press, 1995.

[224] W. Smyth. Computing periodicities in stringsâĂŤa new approach. In *16th Australasian Workshop on Combinatorial Algorithms*, pages 481–488. Citeseer, 2005.

[225] W. Smyth. Computing regularities in strings: A survey. *European Journal of Combinatorics*, 2012.

[226] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree preliminary version. In *Combinatorial Pattern Matching*, pages 140–152. Springer, 1998.

[227] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science*, 270(1):843–856, 2002.

[228] B. Tan and Z. Wen. Some properties of the tribonacci sequence. *European Journal of Combinatorics*, 28(6):1703–1719, 2007.

[229] J. Tang, Y. Lin, C. Balbuena, and M. Miller. Calculating the extremal number. *Discrete Applied Mathematics*, 157(9):2198–2206, 2009.

[230] J. Tang, Y. Lin, and M. Miller. New results on ex graphs. *Mathematics in Computer Science*, 3(1):119–126, 2010.

[231] A. Thomason. On finite ramsey numbers. *European Journal of Combinatorics*, 3:263–272, 1982.

[232] A. Thue. Über unendliche zeichenreihen. *Norske Vid. Skrifter I Mat.-Nat. Kl.*, 1:1–22, 1906.

[233] A. Thue. Ueber die gegenseitige lage gleicher teile gewisser zeichenreihen. *Norske Vid. Skrifter I Mat.-Nat. Kl.*, 7:1–67, 1912.

[234] S. Tishchenko. The largest graphs of diameter 2 and fixed euler characteristics. *Fundamentalnaya i Prikladnaya Matematika*, 7(4):1203–1225, 2001.

[235] S. Tishchenko. Maximum size of a planar graph with given degree and diameter. *Electronic Notes in Discrete Mathematics*, 31:157–159, 2008.

[236] E. W. Weisstein. Polylogarithm: From MathWorld–A Wolfram Web Resource, 2012. `http://mathworld.wolfram.com/Polylogarithm.html`.

[237] Wikipedia. Padovan sequence. `http://en.wikipedia.org/wiki/Padovan_sequence`.

[238] O. Wohlmuth. A new dense group graph discovered by an evolutionary approach. In *Workshop ueber wissenschaftliches Rechnen, Shaker Verlag, Aachen*, 1996.

[239] P. Wong. Cages–a survey. *Journal of Graph Theory*, 6(1):1–22, 1982.

[240] A. C. Yao. A lower bound to palindrome recognition by probabilistic turing machines. Technical report, Stanford University, Stanford, CA, USA, 1977.

Appendix

Appendix A

Additional details for the Subtree repeats problem

In this section more details on the solution of the Subtree repeats problem are given. In particular we show the procedures of the preprocessing phase, an example for the unlabelled case and the procedures for solving the labelled version of the problem.

A.1 Preprocessing phase

In this subsection, the procedures of the preprocessing phase are presented.

ALGORITHM COMPUTE-NODE-PARENTS(φ, n)
1: $S \leftarrow \emptyset$;
2: **for** $i \leftarrow 1$ to n **do**
3: **for** $j \leftarrow 1$ to $\varphi[i]$ **do**
4: $r \leftarrow Pop(S)$;
5: $P[r] \leftarrow i$;
6: $Push(S, i)$;
7: **return** Array $P[1 \ldots n-1]$;

ALGORITHM COMPUTE-SUBTREE-HEIGHT(φ, n)
1: $S \leftarrow \emptyset$;
2: **for** $i \leftarrow 1$ to n **do**
3: **if** $\varphi[i] = 0$ **then**
4: $Push(S, 0)$;
5: $H[i] \leftarrow 0$;
6: **else**
7: $r \leftarrow 0$;
8: **for** $j \leftarrow 1$ to $\varphi[i]$ **do**
9: $r \leftarrow \max(r, Pop(S))$;
10: $H[i] \leftarrow r + 1$;
11: $Push(S, r + 1)$;
12: **return** Array $H[1 \ldots n]$;

ALGORITHM FIRST-CHILD(φ, n)
1: $S \leftarrow \emptyset$;
2: **for** $i \leftarrow 1$ to n **do**
3: **if** $\varphi[i] = 0$ **then**
4: $Push(S, i)$;
5: **else**
6: **for** $j \leftarrow 1$ to $\varphi[i] - 1$ **do**
7: $r \leftarrow Pop(S)$;
8: $FC[r] \leftarrow 0$;
9: $r \leftarrow Pop(S)$;
10: $FC[r] \leftarrow 1$;
11: $Push(S, i)$;
12: $FC[n] \leftarrow 1$
13: **return** Array $FC[1 \ldots n]$;

A.2 Example

In this subsection, we show how the proposed algorithm computes all the subtree repeats of the tree in Fig. 4.1.

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$post(t)$	a_0	a_0	a_0	a_1	a_2	a_0	a_1	a_3	a_0	a_1	a_1	a_1	a_0	a_0	a_1	a_2	a_2	a_0	a_0	a_2	a_0	a_0	a_1	a_2	a_4
P	8	5	4	5	8	7	8	25	10	11	12	17	16	15	16	17	25	20	20	25	24	23	24	25	-
H	0	0	0	1	2	0	1	3	0	1	2	3	0	0	1	2	4	0	0	1	0	0	1	2	5
FC	1	1	1	0	0	1	0	1	1	1	1	1	1	1	0	0	0	1	0	0	1	1	0	0	-

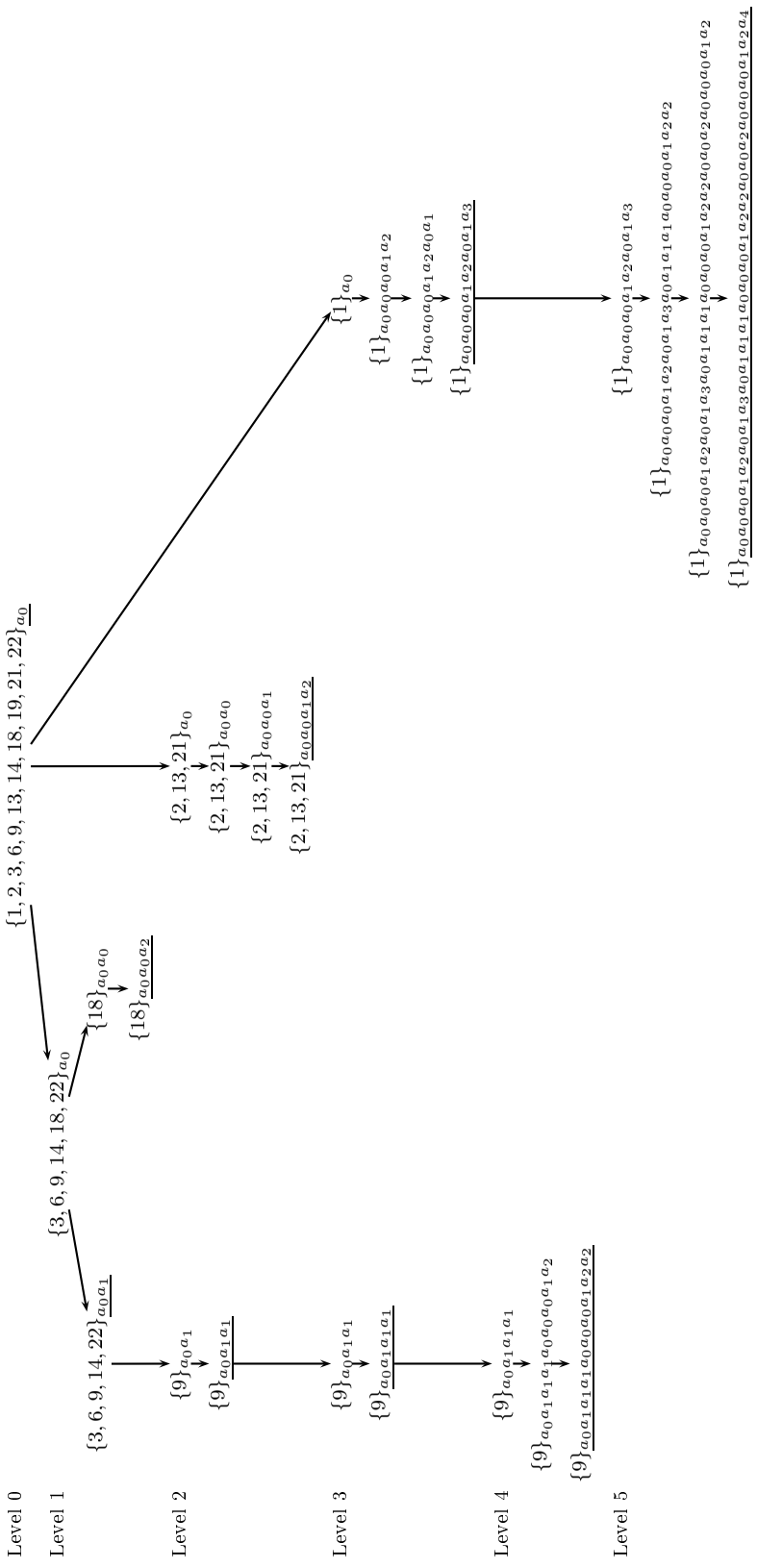
Table A.1: Preprocessing output

index	Sets
0	$\{1, 2, 3, 6, 9, 13, 14, 18, 19, 21, 22\}_1$
1	$\{3, 6, 9, 14, 18, 22\}_1$
2	$\{9\}_2, \{2, 13, 21\}_1$
3	$\{9\}_3, \{1\}_1$
4	$\{9\}_4$
5	$\{1\}_8$

Table A.2: Level array

index	Factors of subtrees
1	a_0
2	$a_0a_0a_2$
3	a_0a_1
4	$a_0a_0a_1a_2$
5	$a_0a_1a_1$
6	$a_0a_0a_0a_1a_2a_0a_1a_3$
7	$a_0a_1a_1a_1$
8	$a_0a_1a_1a_1a_0a_0a_0a_1a_2a_2$
9	$a_0a_0a_0a_1a_2a_0a_1a_3a_0a_1a_1a_1a_0a_0a_0a_1a_2a_2a_0a_0a_2a_0a_0a_0a_1a_2a_4$

Table A.3: Indexing of subtrees


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
{1, 2, 3, 6, 9, 13, 14, 18, 19, 21, 22}$_{a_0}$
{3, 6, 9, 14, 18, 22}$_{a_0}$
{3, 6, 9, 14, 22}$_{a_0a_1}$
{18}$_{a_0a_0}$
{18}$_{a_0a_0a_2}$
{9}$_{a_0a_1}$
{9}$_{a_0a_1a_1}$
{9}$_{a_0a_1a_1}$
{9}$_{a_0a_1a_1a_1}$
{9}$_{a_0a_1a_1a_1}$
{9}$_{a_0a_1a_1a_1a_0}$
{9}$_{a_0a_1a_1a_1a_0a_0a_1a_2}$
{9}$_{a_0a_1a_1a_1a_0a_0a_1a_2a_2}$
{2, 13, 21}$_{a_0}$
{2, 13, 21}$_{a_0a_0}$
{2, 13, 21}$_{a_0a_0a_1}$
{2, 13, 21}$_{a_0a_0a_1a_2}$
{1}$_{a_0}$
{1}$_{a_0a_0}$
{1}$_{a_0a_0a_1a_2}$
{1}$_{a_0a_0a_1}$
{1}$_{a_0a_0a_1a_2a_0a_1a_3}$
{1}$_{a_0a_0a_1a_2}$
{1}$_{a_0a_0a_1a_2a_0a_1a_3}$
{1}$_{a_0a_0a_1a_2a_0a_1a_3a_0a_1a_1}$
{1}$_{a_0a_0a_1a_2a_0a_1a_3a_0a_1a_1a_0a_0a_1a_2a_2a_0a_0a_1a_2}$
{1}$_{a_0a_0a_0a_1a_2a_0a_1a_3a_0a_1a_1a_1a_0a_0a_1a_2a_2a_0a_0a_1a_2a_0a_0a_1a_2a_0a_0a_1a_2a_4}$

A.3 Procedures for labelled ordered ranked trees

In this subsection, the procedures for computing all the subtree repeats of a given labelled ordered rank tree are given.

ALGORITHM PARTITIONING-L(E)

1: **for** $i \in S$ **do**
2: $next = i + \ell_E$;
3: **if** $T[next] \neq 0$ **then**
4: $E_{T[next]} \leftarrow (S_{T[next]} \bigcup \{i\}, \ell_E + TL[next], ac_E - 1)$;
5: **else**
6: $E_{\varphi(post(t)[next])} \leftarrow (S_{\varphi(post(t)[next])} \bigcup \{i\}, \ell_E + 1, ac_E - 1 + \varphi[next])$;
7: **for** every $class$ created in the second step of the above loop **do**
8: **for** $i \in S_{class}$ **do**
9: $next = i + \ell_{E_{class}} - 1$;
10: $E_{label(post(t)[next]),class} \leftarrow (S_{label(post(t)[next]),class} \bigcup \{i\}, \ell_{E_{class}}, ac_{E_{class}})$;
11: **for** every $class$ considered **do**
12: **if** $ac_{class} = 0$ **then**
13: Output S_{class}, ℓ_{class};
14: $sc = sc + 1$;
15: **for** $i \in S_{class}$ **do**
16: $T[i] \leftarrow sc$;
17: $TL[i] \leftarrow \ell[E_{class}]$;
18: Assign level(E_{class});
19: **else**
20: Partitioning(E_{class});
21: **return** Partitioning of E in classes corresponding to next element to be considered;

ALGORITHM SUBTREE-REPEATS-L($post(t), FC, P, H, n, \varphi$)

1: $sc \leftarrow 1$;
2: **for** $i \leftarrow 1$ to n **do**
3: **if** $\varphi(post(t)[i]) = 0$ **then**
4: $E_{label(post(t)[i])} \leftarrow (S_{label(post(t)[i])} \bigcup \{i\}, 1, 0)$;
5: $T[i] \leftarrow sc$;
6: $TL[i] \leftarrow 1$;
7: **else**
8: $T[i] \leftarrow 0$;
9: $TL[i] \leftarrow 0$;
10: **for** every $class$ considered **do**
11: Output S_{class}, 1;
12: $sc = sc + 1$;
13: Assign level(E_{class});
14: **for** $i \leftarrow 1$ to $H[n]$ **do**
15: **while** LA[i] non empty **do**
16: $E \leftarrow pop(LA[i])$;
17: Partition E;
18: **return** Sets of starting positions of Subtrees in post(t) together with their length;